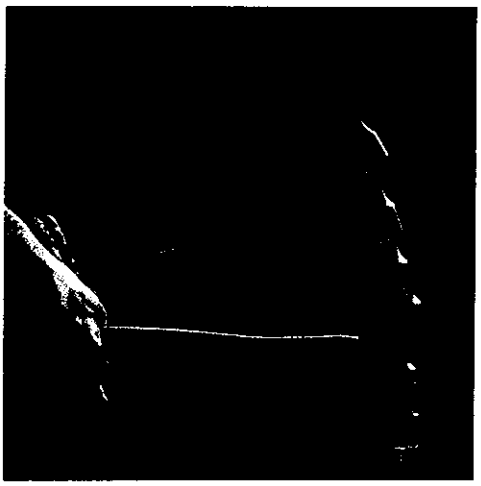


Prototyper™

The Macintosh User Interface Builder with Code Generators for your favorite C and Pascal Compilers.

Copyright

This manual and the software described in it are copyrighted with all rights reserved. Under the copyright laws, neither this manual nor the software may be copied, in whole or part, without the written consent of SmethersBarnes, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others.

© 1987-1990

SmethersBarnes

P.O. Box 639

Portland, Oregon 97207

Telephone: (503) 274-7179

Facsimile: (503) 274-0670

Developed by George R. Cossey.

Limitations on Warranty & Liability

SmethersBarnes warrants that the disk(s) on which this software is recorded is free from defects in material and workmanship, under normal use for 90 (ninety) days after the date of original purchase. If a defect occurs during the 90-day period, you may return the disk(s) to SmethersBarnes for a free replacement. **SmethersBarnes will replace the disk(s) provided that you have previously returned your product registration card to Now Software.** The remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including but not limited to loss of profit, and special, incidental, or other similar claims. The software is licensed and delivered on an "as is" basis. Except for the express warranty set forth above, SmethersBarnes makes no other warranties, either express or implied, by statute or otherwise, regarding the enclosed computer software product, its quality, its performance, its merchantability, or its fitness for any particular purpose.

Macintosh, Imagewriter, LaserWriter, ResEdit, MPW C and MPW Pascal are trademarks of Apple Computer, Inc. THINK C and THINK Pascal are trademarks of Symantec Corporation.

Prototyper is a trademark of SmethersBarnes.

Welcome

Welcome to Prototyper 3.0

Congratulations! SmethersBarnes thanks you for purchasing Prototyper.

Prototyper helps you do in minutes what once took days, weeks, or even months to accomplish with traditional resource editors, paint programs and utilities.

Developed by veteran programmer George R. Cossey, Prototyper was first released in January 1987, embodying a totally new approach to user interface design, prototyping, and software development.

This documentation describes Prototyper 3.0, the latest release. Now Prototyper fits into your development cycle even more closely, with a completely revised architecture for generated source code.

Whether you're a user interface designer or programmer, you'll find dozens of powerful new features in Prototyper 3.0. Yet because this release builds on the existing strengths of Prototyper's simple interface, you'll find these new features easy to learn and use.

New features of Prototyper 3.0

- **Sophisticated new Code Generators**
- **Enhanced screen layout capabilities**
- **C code generation** Prototyper 3.0 now supports THINK C™ 4.0 and Apple Computer's MPW™ C v3.1.
- **Pascal code generation** Prototyper 3.0 supports THINK Pascal™ 3.0, and Apple Computer's MPW™ Pascal v3.1.
- **Iterative development architecture** which allows you to add functionality to your prototype, then revise the interface. Regenerate source code *at any time*, as often as you like — without overwriting your existing custom routines.
- **Enhanced application programmability** Prototyper calls stub routines in your user files at all the places where you're likely to want to get control, letting you easily override its default processing routines. And printing and file I/O are now built-in to the generated source code.
- **Customizable code generation** By placing custom code in new string resources inside Prototyper's Code Generators, you can add your own *uses* and *include* statements to every prototype for which you generate source code — automatically!
- **Custom controls** let you install special code modules giving full control over the behavior and appearance of windows, buttons, toggles and gauges. You can even use Prototyper to design non-Macintosh interfaces!
- **Linking** has been considerably enhanced. Sounds may now be linked to user interaction with your prototype, and linking supports full control of menus, menu items, the enabling and disabling of zones, and a host of other capabilities .
- **Small Icons** Support has been added for 16 by 16 pixel icons, helping to meet the challenges of today's tight constraints on screen real-estate.
- **Palettes** Prototyper 3.0 supports palettes for use in applications which must provide quick access to a variety of tools, particularly useful for prototyping painting and drawing applications.
- **Set location** The positions and dimensions of multiple windows, including dialogs and alerts, can now be displayed simultaneously in Prototyper's Set Location facility, making positioning windows effectively easier, optimizing the use of screen real estate.
- **Other new features** Generated Resource source code now supports both RMaker and Rez, Apple Computer's powerful resource compiler for programmers. The Prototyper Code Generators now implement standard Open... and Save As... file dialogs, as well as Page Setup... and Print... dialogs.

Learning about Prototyper

What is prototyping?

A prototype is a working model of the interface of an application. This model can then be assessed to determine if an optimal design has been achieved. But without a tool for building prototypes, creating one can be almost as difficult and expensive as creating the application itself!

Prototyper 3.0 helps you to develop, test, and communicate your ideas, by providing an interactive example of your interface designs *without* having to spend hundreds of hours writing a program from scratch. With Prototyper 3.0, you can quickly explore as many interface variations as you wish, without getting locked-in prematurely to early approaches. Prototyper's easy-to-learn graphical interface editors let you simply draw the interface attributes you need, and instantly simulate them to refine your design.

Using Prototyper's incredible Code Generation, even non-programmers can produce a stand-alone double-clickable version in seconds flat. And even the least-experienced programmer can add functional extensions to generated code to complete a full-blown custom application in a fraction of the usual time.

Prototypes are most valuable in the critical early phases of software development, but can also assist you in the following areas:

- Testing and communicating alternatives during specification and implementation
- Preliminary beta-testing and market analysis
- Analyzing user/system interactions, and testing software ergonomics
- Measuring the progress and accuracy of development
- Creating documentation and training materials in parallel with software development

Prototyper's new customizable code generation takes the tedious work out of building a Macintosh application skeleton. And with Prototyper's improved source code architecture, you can integrate Prototyper fully into your development cycle.

The bottom line: prototyping increases your potential for success in the identification, design, development, and delivery of software products.

How to use this manual

This manual assumes that you are familiar with at least one Macintosh application, and that you know how to use the Macintosh and the mouse. You should know how to select menus and be familiar with the different types of windows found in Macintosh applications. See your Macintosh owner's manual if you need more information.

It helps to be familiar with Apple Computer, Inc.'s *Human Interface Guidelines*. Originally published with *Inside Macintosh*, and now available separately from Addison-Wesley, it's a valuable exploration of the core concepts and user-centered design philosophy fundamental to the success of the Macintosh graphical interface.

Whether you're a user interface designer or a part-time programmer, this manual will help you make the best use of your time, experience, and Prototyper's powerful development features.

The following briefly describes each chapter:

- ***Getting Started*** — An introduction to Prototyper, including how to install Prototyper, an overview of Prototyper's basic components, and a short tutorial.
- ***Creating Windows*** — Detailed instructions for creating standard Macintosh windows, dialogs, alerts, and demo windows. Describes how to position, resize, change type, delete, import/export, and add other attributes to windows.
- ***Creating Zones*** — Instructions for creating, resizing, moving, grouping, aligning, deleting, and otherwise manipulating Zones in Prototyper's window Worksheets. Shows how to use the Icon Editor to create your own icons, and the Picture Editor to import graphics into your Prototype.
- ***Creating the Menu Bar*** — How to use the Menu Editor to create menus, menu items, hierarchical menus, and menu linking techniques.
- ***Linking*** — Describes how you can link features of your prototype to support user interaction.
- ***Loading Resources & Pritning***— Illustrates how to load existing Macintosh interface resources into your prototype from other applications or Prototyper files — including menu bars, windows, dialogs, alerts, icons, and pictures. A short description of Prototyper's printing features.
- ***Generating Code & Resources*** — An overview of code generation using C and Pascal Code Generators. How to generate code for your chosen compiler, and then to compile the code in your preferred development environment.
- ***Prototyper Menu Reference***— A reference for Prototyper's menus and menu items.
- ***Glossary*** — Prototyper and Macintosh user interface components are described for your reference.

Table of Contents

1. Getting Started

Installing Prototyper 1-1
What your package includes 1-2
Filling out your Registration card 1-3
Prototyper Fundamentals 1-4
Prototyper Tutorial 1-7

2. Creating Windows

Window Characteristics 2-2
Window Behavior 2-4
Window Types 2-5
Dialog Boxes 2-9
Alerts 2-11
Demo Windows 2-13
Creating a Window 2-14
Window Worksheets 2-16
Resizing the Worksheet 2-17
Specifying Additional Window Attributes 2-19
Changing Window Type 2-25
Setting Window Location 2-26
Removing Windows 2-28
Duplicating Windows 2-29
Exporting and Importing Windows 2-30
Simulating Windows 2-31
Simulating entire Prototypes 2-32
Editing Existing Windows 2-32

3. Creating Zones

Manipulating Zones 3-2
Grids 3-2
Creating Zones 3-3
Zone Coordinates 3-4
Resizing Zones 3-5
Moving Zones 3-6
Aligning Zones 3-8
Locking Zones 3-9
Description of Zone Types 3-10
- Buttons 3-10
- Icons 3-12
- Small Icons 3-15
- Check Boxes 3-18
- Radio Buttons 3-20
- Scroll Bars 3-22
- Pop-up Menus 3-24
- Static Text Zones 3-26
- Edit Text Zones 3-27
- Manipulating text for Static or Edit Text Zones 3-28
- Lists 3-29
- Rectangles 3-31
- Lines 3-33
- Pictures 3-34
- Plugin Buttons 3-37
- Plugin Toggles 3-38
- Plugin Gauges 3-39
- Palettes 3-40
- HotSpots 3-42

4. Creating Menus

Menu Editor 4-2
- Editing Menus 4-3
- Standard Macintosh Menus 4-4
- Adding New Menu Titles to the Menu Bar 4-5
- Editing Menu Titles 4-6
- Deleting Menus from the Menu Bar 4-7
- Rearranging Menus 4-8
- Cutting, Copying and Pasting Menus 4-9

Commands 4-10
- Adding New Commands 4-11
- Disabling Commands 4-12
- Adding Separator Lines Between Commands 4-13
- Changing Font Styles for Commands 4-14
- Designating Commands to be Initially Checked 4-15
- Designating Command Keys 4-16
- Designating Icons for Commands 4-18
- Editing Commands 4-19
- Deleting Commands from Menus 4-20
- Rearranging Commands 4-21
- Cutting, Copying and Pasting Commands 4-22

Hierarchical Menus 4-23

Linking Menu Commands 4-25

5. Linking

The Linking Dialog 5-2
- Linking Dialog Icon Buttons 5-3
- Linking From a Menu Command 5-7
- Linking From Zones 5-8
- Creating a Link Map 5-9

6. Loading Resources & Printing

What are Resources? 6-2
Loading Resources 6-3
Printing 6-6
- Printing Menus and Windows 6-6
- Printing to a document 6-6

7. Code Generation

Generating Source Code 7-2
- Saving a prototype 7-3
- Opening the source code generator 7-3
- Selecting a target development system 7-4
- Generating source code and resources 7-5
- Compiling Generated Source Code 7-7
- Using THINK Pascal v3.0 by Symantec 7-8
- Using THINK C v4.0 by Symantec 7-13
- Using MPW Pascal v3.1 by Apple Computer, Inc. 7-18
- Compiling with MPW C v3.1 by Apple Computer, Inc 7-22

Formatting & Other Code Generation Options 7-25
- Pascal Code Generator 7-25
- C Code Generator 7-28

Trouble Shooting 7-31
- Syntax Errors 7-31
- "Code Segment Too Large" errors 7-31
- Link Errors 7-31
- "File Not Found Errors" 7-32
- Regeneration Problems 7-32
- Type Checking Problems in THINK C 7-32
- Windows with identical names 7-32
- Logic Errors 7-33
- Alerts 7-33
- Crashes 7-33
- Default Buttons 7-33
- Demo Windows don't appear 7-33

Regenerating Source Code 7-34
Regenerating source code with THINK C or Pascal 7-35
Replace Core Source Files 7-35
Replace Core Resource Files 7-35
Discard new User Resource File 7-36
Update User Source Files 7-36
Recompile your Project 7-36
Regenerating source code with MPW C or Pascal 7-38
Trouble Shooting Regeneration Problems 7-40

8. Inside Generated Source Code

Core Files 8-2
Core Resources 8-2
XXX.RSRC — Full Core compiled resources 8-2
P_XXX.RSRC — Partial Core resources 8-2
P_RM_XXX.R — Partial Core RMaker text file 8-3
P_Rez_XXX.R — Partial Core Rez text file 8-3
Core Source Files 8-4
P_Common_XXX.Pas / P_Common_XXX.c & P_Common_XXX.h — Core Common file 8-4
PPrint_Common_XXX.Pas / PPrint_Common_XXX.c & PPrint_Common_XXX.h — Core common printing 8-8
PUtils_XXX.Pas / PUtils_XXX.c & PUtils_XXX.h— Core utilities 8-9
Main_XXX.Pas / Main_XXX.c — Core main program 8-11
P_W_XXX.Pas / P_W_XXX.c — Core window handler 8-12
P_IMenus_XXX.Pas, or P_IMenus_XXX.c — Core menubar initialization 8-14
P_DoMenus_XXX.Pas / P_DoMenus_XXX.c — Core menubar handler 8-15
PI_XXXMenu.Pas / PI_XXXMenu.c — Core menu initialization 8-15
PDo_XXXMenu.Pas / PDo_XXXMenu.c — Core menu event handling 8-15
P_MD_XXX.Pas / P_MD_XXX.c — Core Modeless dialog handler 8-16
P_D_XXX.Pas / P_D_XXX.c — Core Modal dialog handler 8-17
P_A_XXX.Pas / P_A_XXX.c — Core Alert handler 8-18
User Files 8-19
User Resources 8-19
U_XXX.RSRC — User Partial resource file 8-19

-XXX.R — User Partial RMaker source file 8-19
XXX.Rez — User Partial Rez source file 8-19
User Source Files 8-20
Events_XXX.Pas / Events_XXX.c — User Event override routines 8-20
Common_XXX.Pas / Common_XXX.c — User common file 8-20
Utils_XXX.Pas / Utils_XXX.c — User stub utilities 8-21
InitExit_XXX.Pas / InitExit_XXX.c — User Startup and exit routines 8-22
XXX.Pas / XXX.c — User Window handler 8-22
InitMenuXXX.Pas / InitMenuXXX.Pas — User menu initialization 8-23
DoMenus_XXX.Pas / DoMenus_XXX.c — User menubar handler 8-23
XXXMenu.Pas / XXXMenu.c — User Menu list handler 8-23
XXX.Pas / XXX.c — User Modeless dialog handler 8-24
XXX.Pas / XXX.c — User Modal dialog handler 8-25
XXX.Pas / XXX.c — User Alert handler 8-25

9. Using the Plugin Installer

Plugin Resources – An Overview 9-2
Custom Controls – 'CDEF' resources 9-2
Custom Windows – 'WDEF' resources 9-3
Custom Sounds – 'snd ' resources 9-4

10. Prototyper 3.0 Menus

File Menu 10-1
Edit Menu 10-2
Layout Menu 10-3
Info Menu 10-4
Special Menu 10-5
Font Menu 10-6
Style Menu 10-6

11. Glossary 11-1

12. Index 12-1

Getting Started

1. Getting Started

Installing Prototyper 1-1
What your package includes 1-2
Filling out your Registration card 1-3
Prototyper Fundamentals 1-4
Prototyper Tutorial 1-7

Getting Started

This chapter will help you to learn Prototyper at your own pace.

You will first install Prototyper and the Code Generator applications. A short explanation of the basics of Prototyper follows, including a quick tutorial.

Installing Prototyper

Before installing Prototyper 3.0:

1. **Check that your Prototyper package is complete.**
2. **Check that your Prototyper master disks are locked. The write-protect tabs should be open.**
3. **Note down the product serial number from the back of the Program disk, and keep it in a safe place.**
4. **Consult the "Readme" document on disk 1.**
5. **Check that you have the required versions of the Finder and System.**

   Prototyper requires version 6.0.3 of the System and Finder, or later.
6. **Backup all Prototyper disks, and store the originals in a safe place.**
7. **Copy Prototyper onto your hard disk.**
8. **Fill out and mail your Registration Card.**

What your package includes

The Prototyper 3.0 package contains:

- Prototyper 3.0 Program disk
- Prototyper 3.0 Pascal Generator and C Code Generator disks
- This manual
- Registration Card

If you did not receive all these items, or if you have any other packaging problems, please call SmethersBarnes at (503) 274-2800 for assistance.

"Readme" document

Please check this file on the Prototyper 3.0 Program disk for any additions, compatibility notes, or last-minute changes to Prototyper that occurred after this manual was printed.

What you need to get started

To use Prototyper 3.0, you need a minimum system configuration of a Macintosh Plus, two 800k floppy disk drives, Finder version 6.1.4, and System version 6.0.4 (included on the Prototyper disk). A hard disk is highly recommended.

You should be familiar with at least one Macintosh application - the examples in this manual refer to a "typical" Macintosh painting or drawing program. No prior programming experience is required, though it would be helpful for getting maximum benefit from Prototyper's Code Generation capability.

Making backup copies

Your license agreement authorizes you to make one set of backup copies of Prototyper 3.0. Make a backup copy of each disk now, then store the originals in a safe place. Keep the original disks locked to prevent accidental damage.

If you are using a hard disk, install Prototyper, then store the locked original disks and backups in a safe place. If you don't have a hard disk, make sure you use only your (unlocked) backup disks as the working copies of Prototyper, not your original Prototyper disks.

Installing on a hard disk

You may already have an earlier version of Prototyper. If the previous version or any folder is named "Prototyper", you should rename it now so that you don't accidentally overwrite it while installing Prototyper 3.0.

To install Prototyper on a hard disk or 800k drive:

1. **If it's not already on, switch on your computer.**
2. **Create a new folder labeled "Prototyper" on your hard disk.**
3. **Insert the Prototyper 3.0 disk into any drive.**
4. **Copy the contents of the Prototyper disk into the Prototyper folder you just created.**

   If you are already using the appropriate system software, do not copy the Prototyper System Folder to your disk.
5. **Eject the Prototyper disk, and insert the Pascal and C Code Generator disks.**
6. **Copy the contents into the Prototyper folder.**
7. **Store the original disks in a safe place.**

   If you have two 800k drives and no hard disk, do not install Prototyper. Use the backup copies you made earlier as working copies.

   The first time you open Prototyper, you must personalize your copy. Enter your name and company name in the appropriate lines. If you do not do so properly, Prototyper will exit to the Finder.

Filling out your Registration card

Fill out your Registration Card and send it to SmethersBarnes as soon as possible. It is important that we have your Registration Card on file. If your package is shipped with bad disks, or you need help operating Prototyper, we will be better able to assist you if we have your registration. This also allows us to notify you when new versions are available.

Your registration number is printed on the back of the Prototyper disk. If you are a previous owner of Prototyper, and this version of Prototyper is an upgrade, please use your original registration number in any communication with SmethersBarnes.

Prototyper Fundamentals

This section explains the basics of constructing the characteristic Macintosh desktop interface with Prototyper. We assume that you already understand how to use the Macintosh and therefore are familiar with its interface and software components.

First, we begin with an overview of Prototyper's main features, including the Prototype Window and its buttons, the help file, and Show Additional Notes.

The Prototype Window

Your prototype is controlled from the Prototype Window, which displays its windows, dialogs, and alerts, and allows you to create new windows or edit menubars at any time. By clicking the menu, window, or demo window buttons located across the top of the Prototype Window, you can build a menu bar, or design any number of different windows, dialog boxes, alerts, or demo windows (which simulate scrolling document windows).

Below these buttons, the titles of your windows are displayed in alphabetical order.

Choose Open from the File menu, or double-click a title in the prototype window to edit a window's contents or change its name.

Menu Button

Clicking the menu button opens the Menu Editor. The Menu Editor lets you build the menu bar, menus, and commands for each prototype. The Menu Editor automatically includes the standard Apple, File, and Edit menus and their usual menu items. You may use or delete these menus, as well as specifying new menus and menu items, (though you should remember that Apple Computer's Human Interface Guidelines stipulate that these first three menus should be consistent across all Macintosh applications).

You can also assign menu item command keys, disable menu items, assign icons to menu items and link menu commands to open or close windows. Each time you click the menu button, the Menu Editor appears, letting you edit the menu bar of your prototype. The Menu Editor is described in full in a later section.

Window Button

Clicking the window button displays a dialog box, letting you select a window type to create. You can choose from all of the standard Macintosh windows, dialog boxes and alerts. If you have used Prototyper's Plugin Installer to add a custom window (WDEF resource) to Prototyper, you will also see an icon representing each custom window.

Clicking on the window button displays this dialog box, letting you create a new window at any time.

Demo Window Button

To simulate a fully scrolling Paint or Text document window, use Prototyper's "demo" window facility. Do this by clicking the demo window button.

You can create a fully scrolling document window for demonstration purposes. Select text or a picture from a text or graphics file to load into your prototype's demo window.

Note: Demo Windows are not supported by Prototyper's Code Generators — they are an aid for simulation purposes only.

Prototyper Help File

Prototyper's Help facility provides immediate assistance with menu commands, creating windows, zones, menus, loading resources and many other subjects. Prototyper's Code Generators also include their own Help facilities, accessible from the About Prototyper dialog box.

Help topics are listed alphabetically.

To view the Help File:

1. **Choose About Prototyper from the Apple menu.**
2. **Click the Help button.**

   The Help File window contains two lists. The upper list contains all the topics in the file. Scroll through the upper list and select a topic.

   The topic information appears in the lower list. To move from one topic to another, select a new topic in the upper list.
3. **Click Cancel or press return when you are finished with Help.**

Show Additional Notes

Prototyper provides explanatory notes as you create your prototype. These notes describe the current window or zone and its use in a Macintosh application. If you wish to skip these notes, turn this feature off.

To show additional notes:

1. **Choose Preferences from the Special Menu.**
2. **Click the check box for Show Additional Notes.**
3. **Click OK or press the return key to cancel.**

Prototyper Tutorial

If you need information on performing a particular function in Prototyper, refer to the table of contents to go directly to the appropriate page.

Using a sample prototype, this tutorial teaches how to create and modify a zone, how to create a window, and how to designate a window to open when the prototype starts up. Next, you learn how to build a menu for your prototype, how to load the resources for a dialog, and how to link a menu command to open the dialog.

The sample prototype is re-used in Chapter 7, Generating Code, where source code generated for the sample is loaded into each of the development systems and compiled into a standalone application.

Starting the Tutorial:

1. **Double-click the folder titled "Tutorial" from the Finder.**
2. **Double-click the file titled "Sample Prototype".**

   Prototyper starts up, and the prototype window is opened.

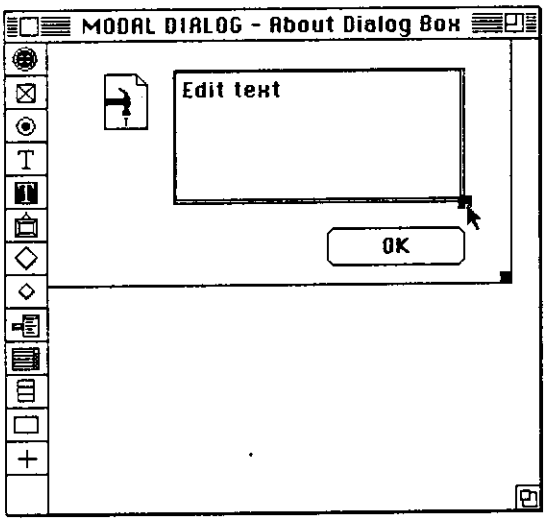
Step 1: Open a Window

1. **Select the "About Dialog Box" entry in the Prototype Window**
2. **Choose Open from the File menu, or double-click "About Dialog Box".**

   This displays the worksheet containing a dialog box entitled "About Dialog Box". The dialog box contains an icon zone and a button zone. Now, you will create a static text zone and enter your name.

Step 2: Create a Zone

1. Select the static text tool from the palette.
2. Draw a rectangle next to the icon as shown.



3. Choose Zone Info from the Options menu, or double-click anywhere in the static text zone (the rectangle you just drew.)
4. Enter "Sample Prototype".
5. Press Return.
6. Enter the word "By", followed by your name.
7. Click OK.
8. Choose any font style from the Style menu.

Step 3: Simulate the Window

1. **Choose QuickLook from the Special menu.**

   QuickLook provides a quick simulation of the current worksheet. Choosing QuickLook causes Prototyper to switch to its simulator, which simulates the dialog box as if it was a part of a real Macintosh application.

2. **Click the OK button in the dialog box.**

3. **Choose Quit from the prototype's File menu.**

   This returns you to Prototyper.

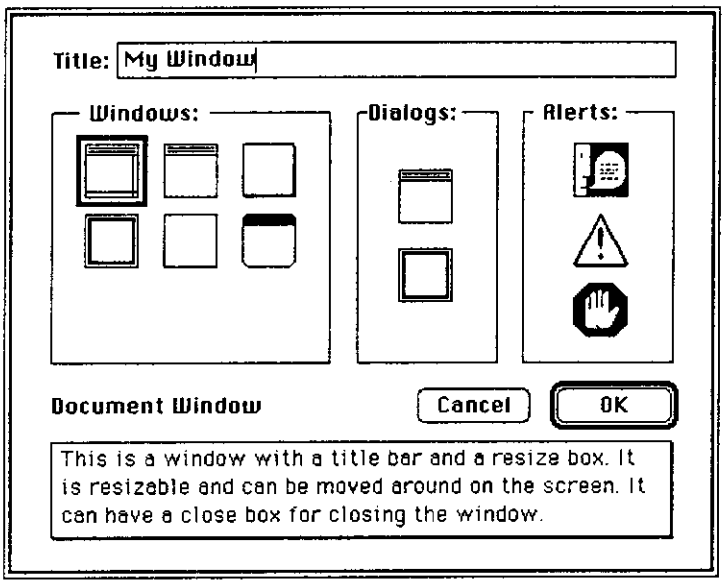
   Note: When creating a prototype, if you have accidentally eliminated the Quit command from the menu bar, or created a modal dialog box without an exit button, the simulator can ALWAYS be exited by pressing the Command Key and typing a (.) period.

4. **Close the worksheet for the dialog box.**

Step 4: Create a Window

1. **Click the Window button in the prototype window.**

   You can select any of the standard Macintosh window types. For now, use the default window: the standard document window.



2. **Enter "My Window" for the title.**
3. **Click OK.**

   The worksheet for "My Window" is displayed.

4. **Choose Window Info from the Info menu.**
5. **Click the check box for Open at program start-up.**
6. **Click the check box for Full screen sized window.**
7. **Click OK.**
8. **Close the worksheet.**

Step 5: Simulate the Prototype

1. **Choose Run from the Special menu.**

   Run simulates the entire prototype. Prototyper's own menu bar is replaced by the menu bar from your prototype. The window you designated to open at program start-up opens. The interface of the prototype can now be demonstrated.

   Notice the window "My Window" looks and feels like a "real" standard document window. It can be moved on the screen and resized.

2. **Choose About from the prototype's Apple menu.**

   The "About dialog box" appears, with your name as the designer. Click the OK button.

3. **Choose Quit from the prototype's File menu.**

   This returns you to Prototyper.

Step 6: Load Resources

1. **Choose Load resources from the Special menu.**
2. **Open the Sample Dialog Resources file.**

   A dialog box appears, showing the types and number of resources available for loading.
3. **Click the check box for Dialogs.**
4. **Click the Load button.**

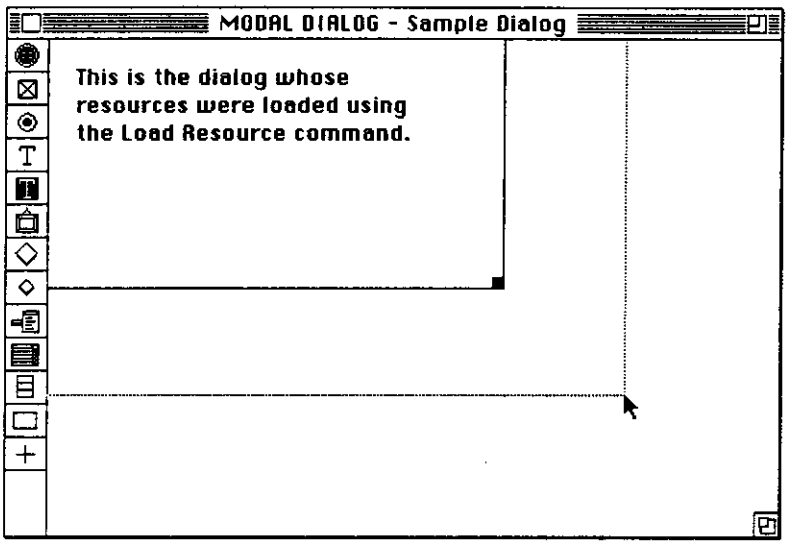
There is one dialog box and a menu bar available. Load only the dialog box. When Prototyper is through loading resources, the title is listed in the Prototype Window.

The resources have been loaded, and the dialog box is now a part of your prototype. For more information, refer to the section on loading resources.

Step 7: Resizing the Worksheet

Now that you have loaded the Sample Dialog, you can resize it and manipulate it as you wish.

1. **Select the "Sample Dialog" entry in the prototype window and choose Open from the File menu, or double-click the "Sample Dialog" entry.**
2. **Press the worksheet's size box and drag the worksheet larger as shown.**



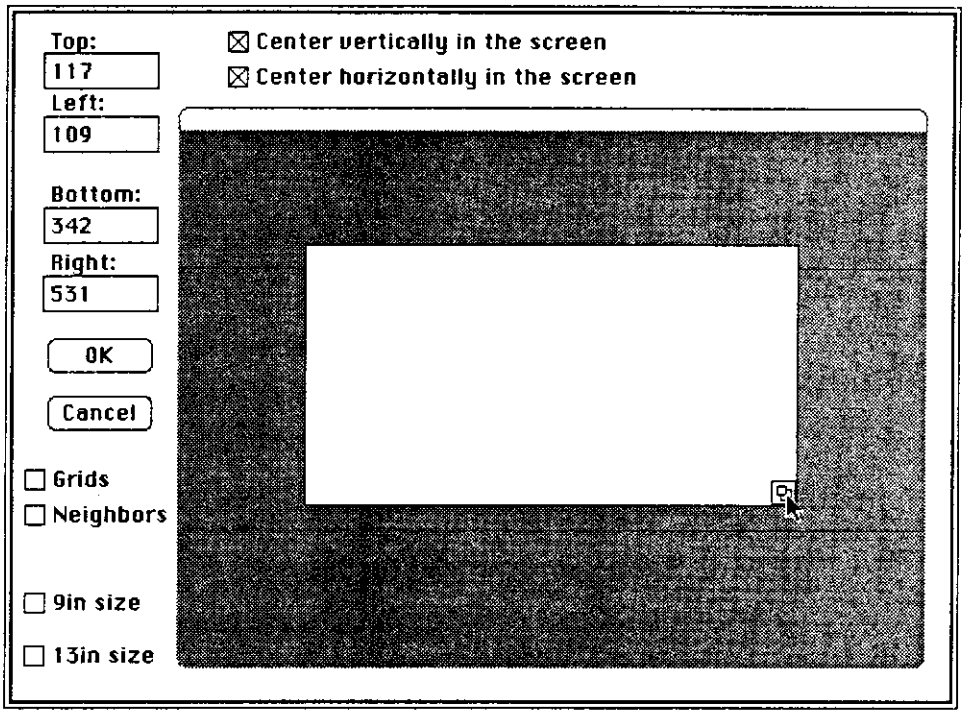
3. **Press the handle in the lower right hand corner of the dialog. Drag the outline to the new size as shown.**

Step 8: Set the Dialog's Location

The initial location of a window on the screen is specified with the Set Location Dialog. Prototyper automatically defaults dialogs vertically to the standard position, the upper 1/3 of the screen. Because you have changed the original size of the dialog, you should also adjust its location.

1. **Choose Set Location from the Layout menu.**

   A dialog appears with a miniature Macintosh screen. On the screen is a white box representing the Sample Dialog.



2. **Click the Center Vertically check box**
3. **Click the Center Horizontally check box.**

   You have now set the window's location to be centered on the screen horizontally and vertically. If you are using a large monitor, the window will be positioned accordingly.

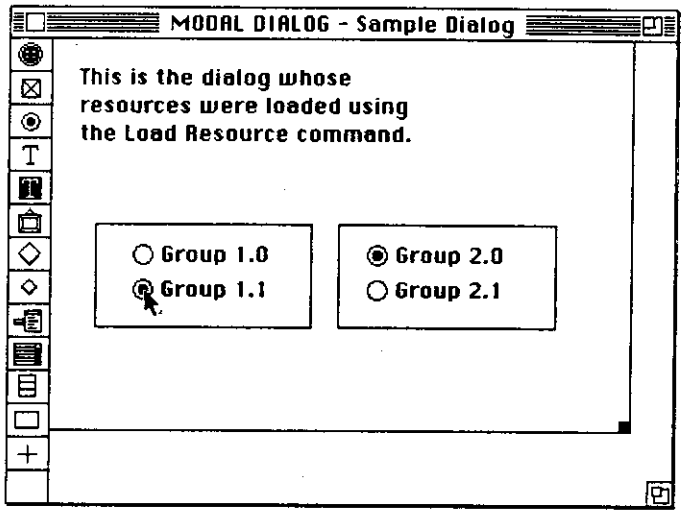
   Note: Notice that the coordinates of the window are updated to represent the new pixel coordinates.

4. **Click OK.**

Step 9: Add Groups of Radio Buttons

Now, you will add two groups of two radio buttons, and draw rectangles to enclose them.

1. **Select the radio button tool from the palette.**
2. **Draw a small, long rectangle as shown.**



3. **Choose Duplicate from the Edit menu and make three more radio buttons.**
4. **Drag the radio buttons into groups as shown.**
5. **Double-click each radio button in the group on the left, changing their titles to Group 1.0 and Group 1.1.**
6. **Click the appropriate setting for Group 1.**
7. **Double-click each radio button in the group on the right, changing their titles to Group 3.0 and Group 2.1.**
8. **Click the appropriate setting for Group 2.**
9. **Select the rectangle tool from the palette.**
10. **Draw a rectangle around each group of radio buttons as shown.**
11. **A rectangle defaults to being invisible, so double click on the rectangles and change their border width to make them visible, and un-check the HotSpot checkbox.**

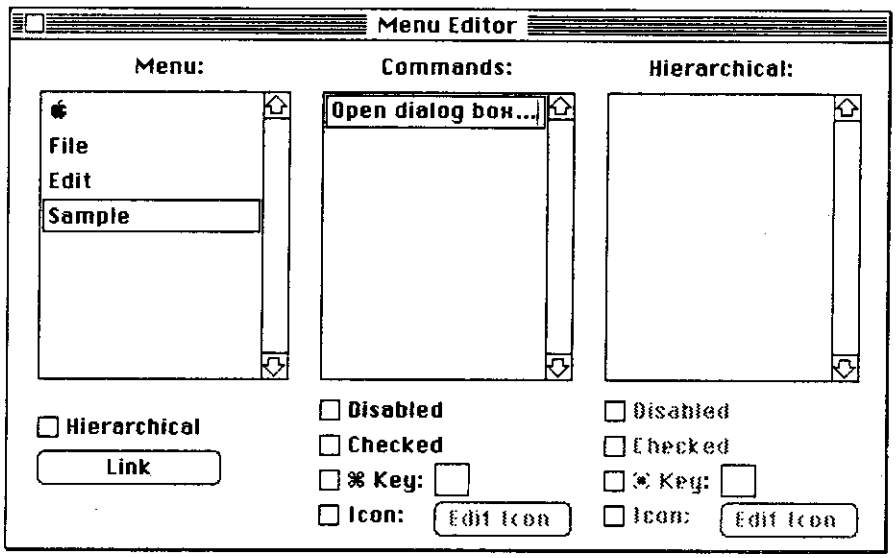
Step 10: Simulate the Dialog

1. **Choose QuickLook from the Special menu.**
2. **Click the radio button for each group.**

   Notice that each group of radio buttons works independently of the other.
3. **Type Command-Period to exit simulation.**
4. **Close the worksheet for the Sample Dialog.**

Step 11. Create a Menu and Command

1. **Click the Menu Button in the Prototype Window.**
2. **Click on the open space after the Edit menu title in the Menu column. Enter "Sample"**



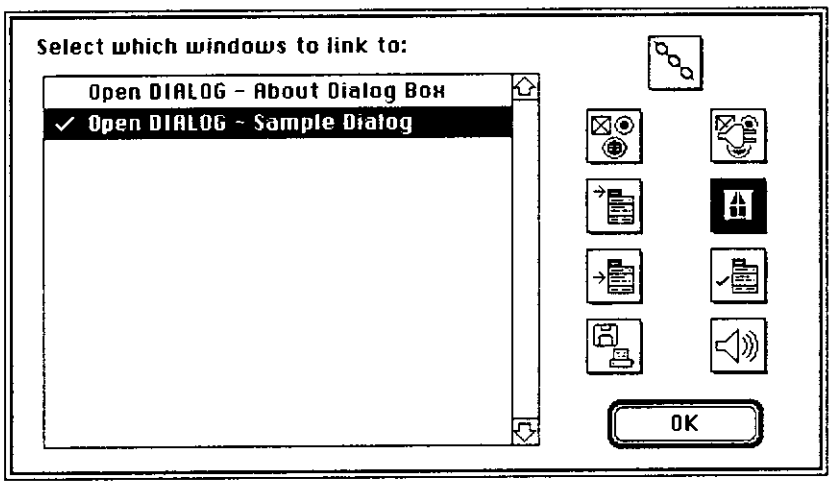
3. **Click on the first open space in the Commands column.**
4. **Enter "Open dialog box..."**

   Make sure you include the ellipsis character (...) after the command. This Macintosh menu convention signifies that a dialog box will appear, with additional information to select from. Press Option-semicolon (;) to get an ellipsis.

Step 12: Link a Command to open a window

1. **Click the Link button.**

   The Link dialog box appears, listing the links that this command executes. Link the command to open a window.



   There are no other links tied to this menu command yet, so the list is empty.

2. **Click the window icon in the Link dialog.**
3. **Select "Sample Dialog".**
4. **Click OK.**

Step 13: Simulate the Menu

1. **Press on the menu title "Sample" on the right in the main Prototyper menu bar.**

   When the Menu Editor is open, the selected menu title, along with the commands in the menu, appears in simulation to the right of the main Prototyper menu bar. At any time while working in the Menu Editor, you can press the simulated menu to display its commands. Each time you select a new menu in the Menu Editor, it appears in the Prototyper menu bar, replacing the previously displayed menu.

2. **Close the Menu Editor**

Step 14: Simulate the prototype

1. **Choose Run from the Special menu to run your prototype.**
2. **Choose Open dialog box from the Sample menu to display the dialog box.**
3. **Choose Quit from the prototype's File menu to return to Prototyper.**

Creating Windows

2. Creating Windows

Window Characteristics 2-2
Window Behavior 2-4
Window Types 2-5
Dialog Boxes 2-9
Alerts 2-11
Demo Windows 2-13
Creating a Window 2-14
Window Worksheets 2-16
Resizing the Worksheet 2-17
Specifying Additional Window Attributes 2-19
Changing Window Type 2-25
Setting Window Location 2-26
Removing Windows 2-28
Duplicating Windows 2-29
Exporting and Importing Windows 2-30
Simulating Windows 2-31
Simulating entire Prototypes 2-32
Editing Existing Windows 2-32

Creating Windows

All Macintosh software, from the simplest desk accessory to the most complex application, is based on windows. Windows display user data, and allow a broad set of methods for interacting with the data. Multiple windows can be simultaneously displayed, with the active window in the foreground, overlapping inactive windows. Active windows may include scrollbars, grow boxes or zoom boxes, which all allow more data to be displayed than can fit on the screen at one time.

Dialogs and alerts are special kinds of windows, which interact with the user in a highly specific and directed fashion, such as allowing parameters to be set, or alerting the user to an unusual situation. For example, a dialog may contain specifications for setting page dimensions for printing, and an alert may warn the user about continuing cautiously in a low-memory situation. Modal dialogs and alerts restrict the user to interacting with them until explicitly dismissed — clicking outside their boundaries yields only a beep. Modeless dialogs allow access to other features of an application while they are displayed on screen.

In this chapter you will learn how to create windows, dialog boxes, alerts, and demonstration windows. Prototyper supports all standard Macintosh window types, and with its Plugin capability, can design interfaces based on completely new window types. In this tutorial, we focus on standard Macintosh window types.

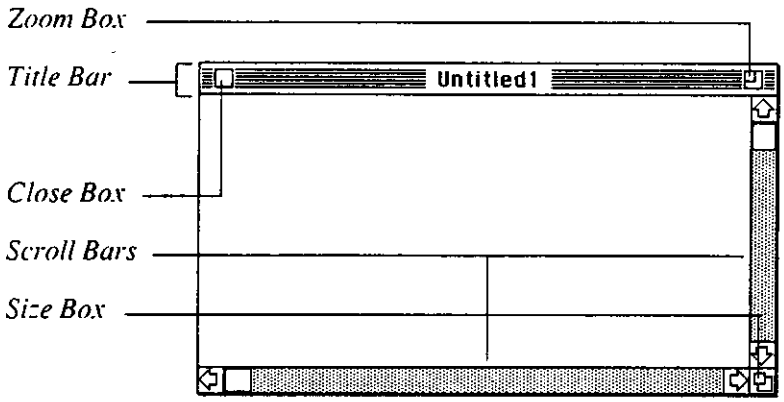
You'll also learn how to resize a window, designate that it be opened at start-up when your prototype is run, and set the location of the window on the screen.

Note: Dialog boxes and alerts look very similar to windows, but cannot include all the types of zones available in windows. In addition, the code generated for dialogs by Prototyper uses the Dialog Manager, instead of the Window Manager, which may lead to some functional limitations that may not be acceptable in all situations: For further explanation of the Dialog and Window Managers, refer to Inside Macintosh.

Throughout the remainder of this chapter, we refer to windows, dialog boxes, alerts, and demo windows collectively as "windows".

Window Characteristics

Windows for Macintosh applications come in various shapes and sizes, each with a special set of characteristics. Windows typically include at least one of the following attributes, but need not contain any of them. For reference, here is a very brief review of these attributes, and their individual functions:



Title Bar

The title bar is displayed across the top of the window, and contains the window's name, the close box, and the zoom box. The title bar displays six horizontal stripes when the window is active, but contains only the title when the window is inactive. Only one window is normally active at a time, and the title bar provides visual feedback to the user to indicate which window is being interacted with. Clicking an inactive window activates it, and deactivates the current window.

Close Box

The close box is a small square located at the left end of the title bar. When you click the close box, the active window disappears.

Size Box

The size box is used for resizing windows. It is located in the bottom right corner of the window. Dragging the size box creates an outline of the window. You drag the outline to the new size and release the mouse button. The window is then redrawn at the new size.

Zoom Box

The zoom box is a small box containing a smaller rectangle at the right end of the title bar. The zoom box expands or contracts a window between two alternate sizes, the user size and the standard size. This standard size may be specially defined by the application, but typically fills the entire screen (regardless of the size of the monitor).

When a window is first opened, it is usually set to the standard size. You may then resize the window as necessary by dragging the size box. Your adjustment then defines the user size. See the Window Manager section of Volume IV of Inside Macintosh for further information on the user size and standard size.

Whenever you click the zoom box, the window "toggles" between the standard size and the user size.

Scroll Bars

Scroll bars change the view of the contents of a document. The scroll box is positioned relative to the grey area exactly in proportion to the current view of the data relative to the size of the whole document. In other words, if the first third of the data is currently visible in the window, the scroll box is positioned in the first third of the grey area.

Scroll bars support several modes of interaction:

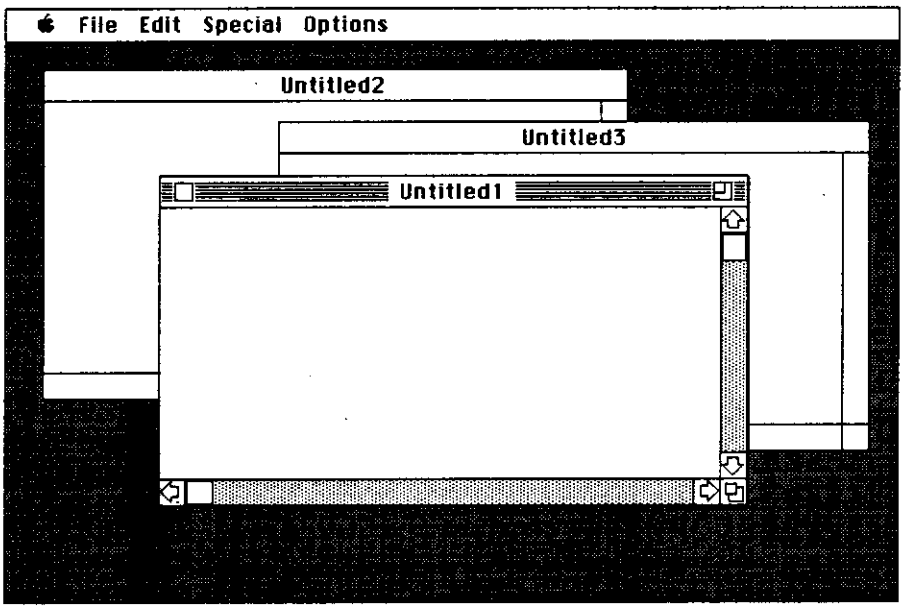
- Dragging the scroll box through the grey area creates an outline of the scroll box, which may be moved anywhere in the grey area. If the cursor moves outside the vicinity of the scroll bar, the outline disappears, and no action is taken when the mouse button is released. Otherwise, the document is immediately scrolled to its new position within the window, in proportion to the new position of the scroll box within the grey area.
- Clicking the grey area between the scroll box and a scroll arrow scrolls the document a window-full at a time, updating the position of the scroll box accordingly. Holding down the mouse button causes continuous scrolling until the end of the document is reached, or the mouse button is released.
- Clicking the scroll arrows scrolls the document within the window, updating the position of the scroll box accordingly. The units of scrolling are defined by the application — in a word-processing application, for example, they would usually be one line of text. Holding down the mouse button causes continuous scrolling until the end of the document is reached, or the mouse button is released.

Window Behavior

All Macintosh windows behave according to certain rules. Although multiple windows may be open on the screen at one time, the user can usually only work on, or interact with, one window at a time. This window is called the active window. All other open windows are considered inactive.

To activate a window, click anywhere in the window. This has two effects:

1. **The window moves to the forefront position, overlapping any other windows which cover the same screen area.**

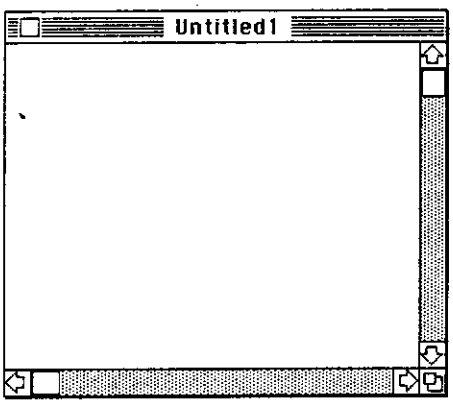


2. **The window's appearance changes. The title bar becomes striped, while the close box, scroll bars, and resize box appear.**

Window Types

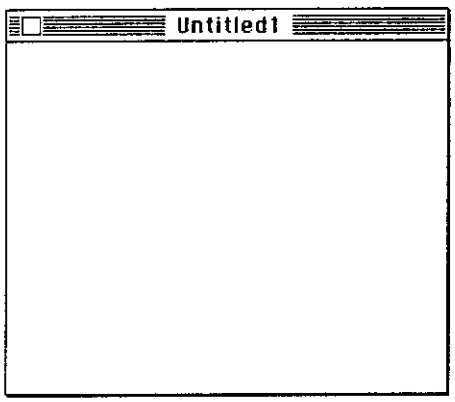
Prototyper supports six standard Macintosh window types; standard document window, document window without a resize box, box window with a drop shadow, alert style window, plain box window, and rounded corner window.

Standard Document Window



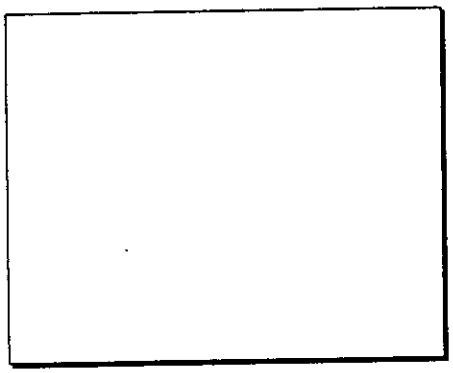
This is a window with a title bar and scroll bars, and is the type of window most commonly used in applications. It is resizable and movable. It often includes a close box for closing the window independently of a menu command.

Document Window w/o Size Box



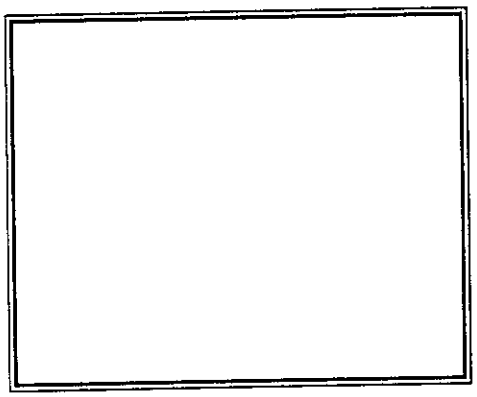
A window without a size box. This is a standard document window which is movable, but cannot be resized. A close box is optional. This window type can be used when there is no possible user benefit from being able to enlarge the window.

Box Window w/shadow



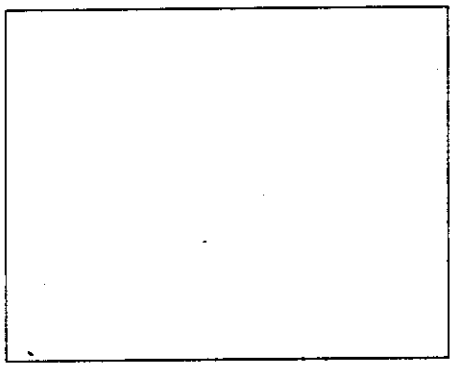
A plain box with no special features except a background shadow to give it the appearance of depth. Its size and screen position are fixed. This type of window is sometimes used for the temporary display of progress data. There is no close box.

Alert Style Window



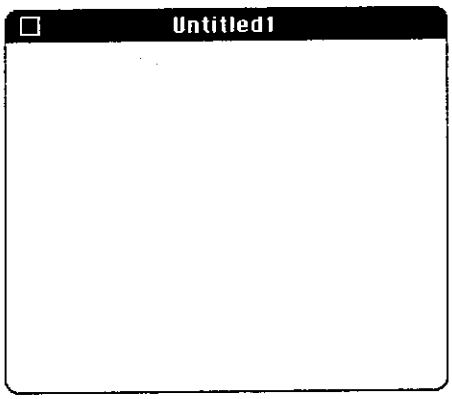
This window looks like an alert, but may have all the zones of a window, and acts like a window. It is neither resizable or movable, and does not include a close box. This window type may be useful to represent Alert "dialog boxes" if you prefer not to use the Dialog Manager.

Plain Box Window



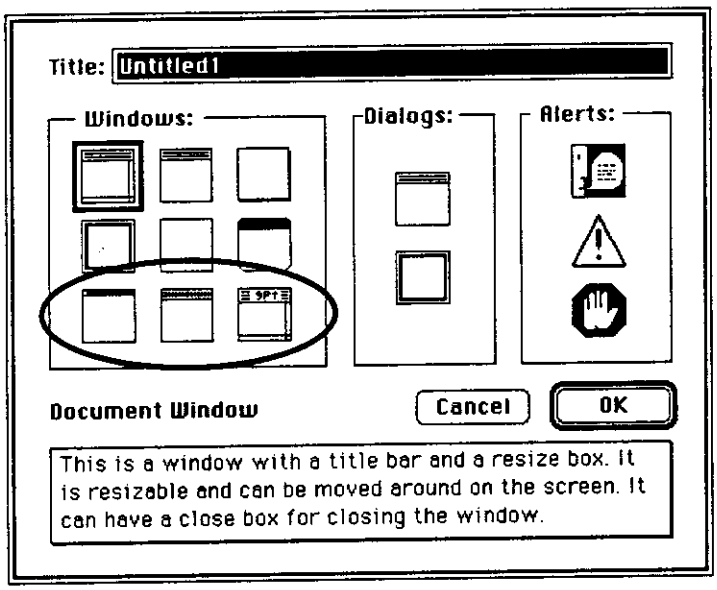
A plain box without any typical window features. Its size and screen position are fixed, and there is no close box. This type of window is sometimes used for the temporary display of progress data, and is relatively rare in Macintosh applications.

Rounded Corner Window



A box with a black title bar and rounded corners, typically used in desk accessories. It is movable, and may have a close box.

Custom Windows



Custom windows that have been plugged in with Prototyper's Plugin installer will appear at the bottom of the windows group. Prototyper will allow up to three to be installed.

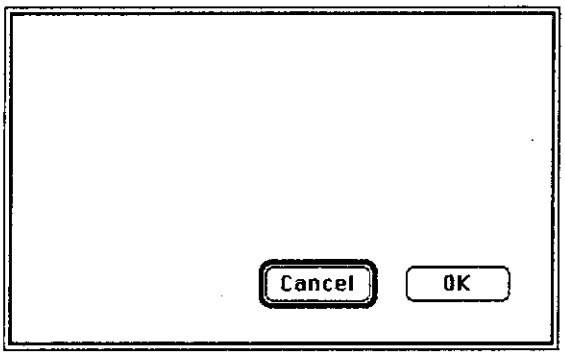
Dialog Boxes

Dialog boxes gather additional information from the user, or provide the user with additional information about an application. They are usually linked to a menu command. Dialog boxes can contain all the zones of a window, except scroll bars.

When dialog boxes provide information to the user, they typically contain static text and at least two buttons; OK and Cancel. An example of a dialog box is the window that appears when About Prototyper... is chosen from Prototyper's own Apple menu. Newly created dialogs automatically include OK and Cancel buttons.

In dialog boxes created with Prototyper 3.0, command keys are fully supported. When a modeless dialog is active, command keys may be used to activate menu choices. Using Command-X, Command-C, and Command-V keystrokes in either modal or modeless dialogs containing an active editable text zone will cut, copy and paste text appropriately within the editable zone.

Modal Dialog Box

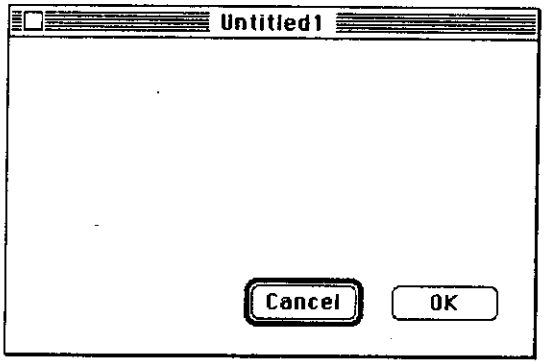


Modal dialog boxes force the user to take action before proceeding, preventing menu selection or other windows being accessed until the dialog box is closed. An example of a modal dialog box is the standard Open dialog box.

Modal dialogs take control of the application from the user, a departure from the normally permissive and flexible Macintosh spirit, and should only be used when absolutely necessary. Newcomers to Macintosh programming often overuse modal dialogs, resulting in an application that is frustrating to use.

Refer to Apple Computer's Human Interface Guidelines for a more detailed discussion of the pitfalls of excessive modality.

Modeless Dialog Box



In contrast with Modal dialogs, above, a modeless dialog box allows the user to interact with other windows while it remains on screen. A modeless dialog box can be closed by clicking the close box or by choosing Close from the File menu. Buttons in modeless dialog boxes should never close the dialog box. An OK button tells the computer to perform the operation, leaving the dialog box open. A Cancel button stops an operation. A familiar example of a modeless dialog box is the Change window used for changing text strings in several popular word processors.

Modeless dialog boxes permit greater flexibility than modal dialogs, since menus and other windows can be reached while a modeless dialog is open. They are therefore much less intrusive than modal dialogs, although they can sometimes be confusing to novice users.

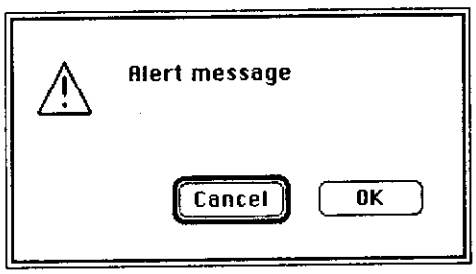
Alerts

Alerts warn that an unusual situation has occurred, and that data may be lost if appropriate action is not taken to correct the problem. There are two kinds of alerts; beeps and alert boxes. Beeps are used for errors that are minor and immediately obvious. Alert boxes resemble a modal dialog box.

Alert boxes typically contain static text explaining why the alert has appeared, and two buttons, OK and Cancel. You must select one or the other to continue. Alerts may only contain buttons, icons, pictures and static text. As an example, a Caution alert may warn you that insufficient memory remains to continue, and suggest that you close a window in order to free some memory.

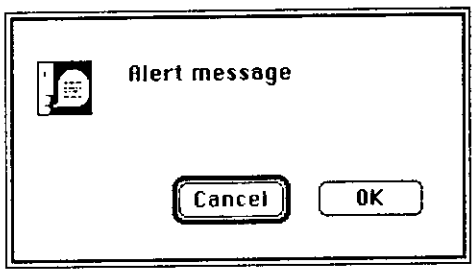
When you create a new alert in Prototyper, its OK and Cancel buttons are created for you, and a static text zone is also created in the window Worksheet for you to use.

Caution Alert



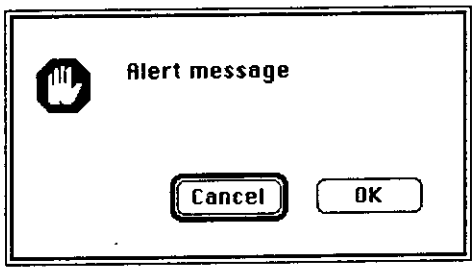
This alert warns that data may be lost if the present course of action is not changed. You are given a choice to continue with your current action or not.

Note Alert



This alert warns of a minor mistake or unusual circumstance. No potential for data loss is ordinarily implied. You may usually respond by simply pressing the OK button.

Stop Alert



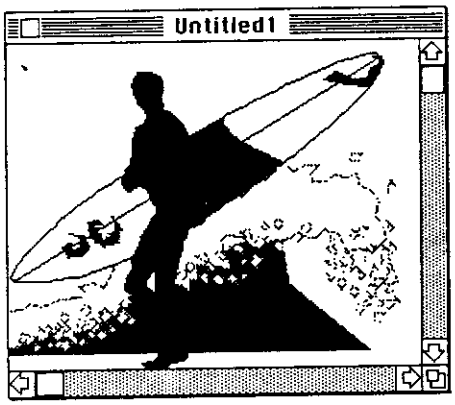
This alert warns of serious problems that may cause an imminent loss of data. Immediate corrective action is requested to prevent data loss.

Demo Windows

Demo Windows are a convenient way to simulate a scrolling window without programming. For discussions and walk-throughs of an interface design, the typical functionality of a graphics or text-editing window can be illustrated.

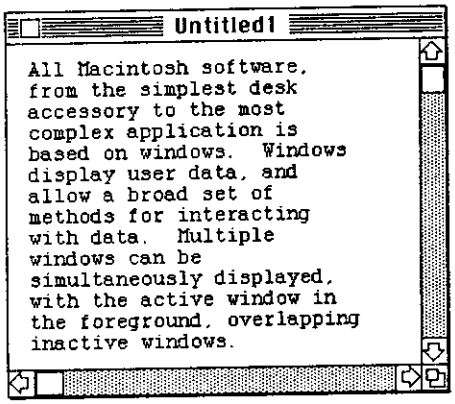
Note: Demo Windows are not supported by Prototyper 3.0 Code Generators.

Picture Window



This demo window uses draw-style or paint-style pictures to conveniently simulate a variety of applications with scrolling graphic capabilities, such as drawing programs, 3D graphics programs, word processors, etc. Scroll bars can scroll the picture within the window horizontally, vertically, or in both directions.

Text Window



This demo window uses a plain text file to display sample text that can be scrolled. You can also specify a maximum horizontal text width. Like Demo picture windows, scroll bars can scroll the text within the window horizontally, vertically, or in both directions.

Creating a Window

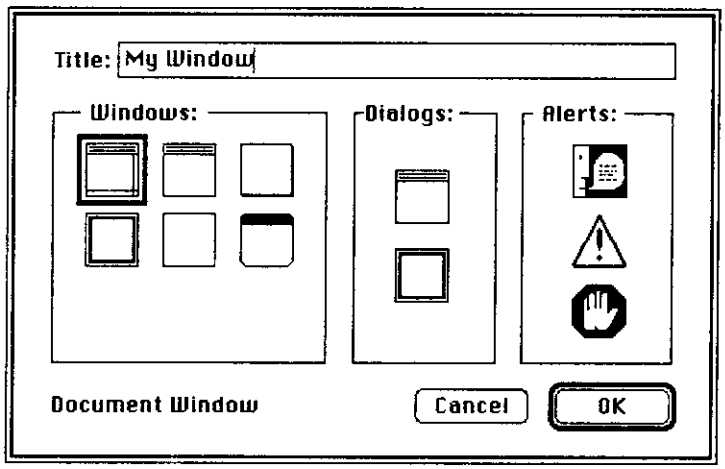
Windows, dialog boxes and alerts are created by selecting the desired window type, then using the Worksheet to add Zones to the window.

To create a window:

1. **Click the window button in the prototype window.**

A dialog box appears with a selection for six standard Macintosh window types, two dialog box types, and three different alert types. Additional window types are displayed if you have installed custom Plugin window definitions with Prototyper's Plugin Installer.

2. **Enter the title for the window.**



The title appears in the window's title bar, in the prototype window, and in generated source code. Of course, if the window type you are using does not support a title bar, the name will not appear in the window.

3. **Select the corresponding window icon for the window you want to create.**

The standard document window is selected by default.

Note: If Show Additional Notes is selected in the Preferences dialog box, Prototyper provides additional notes describing the type of window selected and how that type of window works. Before continuing, click on each window icon to find out how they are generally used.

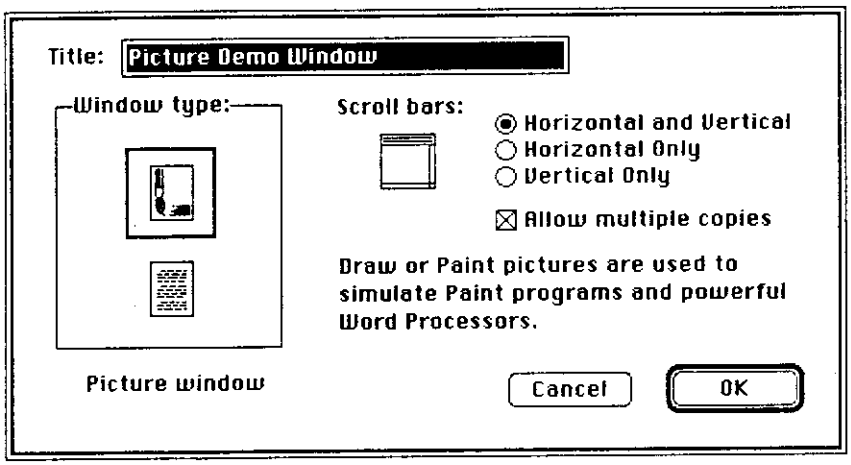
4. **Click OK or press Return.**

To create a demo window:

1. **Click the demo window button in the prototype window.**

A dialog box appears with a selection for two demo window types, and an explanation of the use of demo windows.

2. **Enter the title for the window.**



This title appears in the window's title bar and in the prototype window.

3. **Select the corresponding text or picture icon for the window's scrolling contents.**

The picture demo window is selected by default.

4. **Click OK or press Return.**

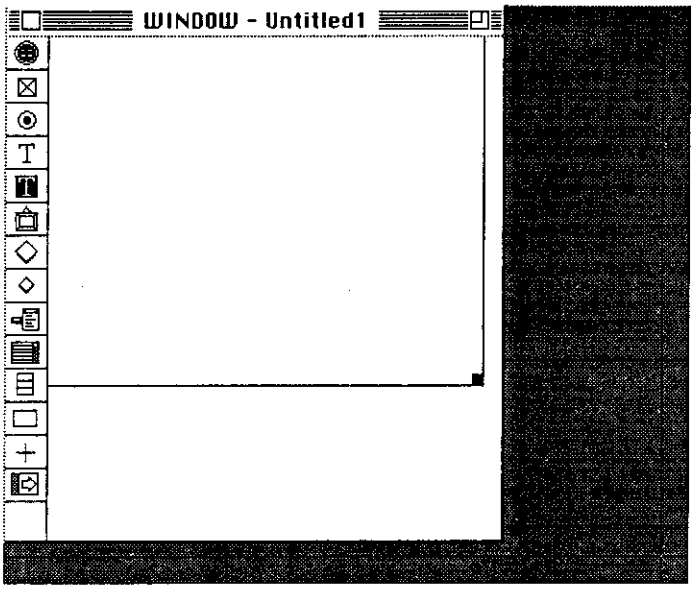
Window Worksheets

The Worksheet and its zone palette let you design a prototype window and its contents. While the Worksheet is open, the size or location of the window can be changed, additional window attributes can be specified, and zones can be created using the palette.

Note: You can learn how to use Worksheets in the Creating Zones chapter.

Resizing the Worksheet

The Worksheet must be big enough to allow you to design your window or dialog. To give yourself more room, drag the Worksheet's Size box to expand the Worksheet.

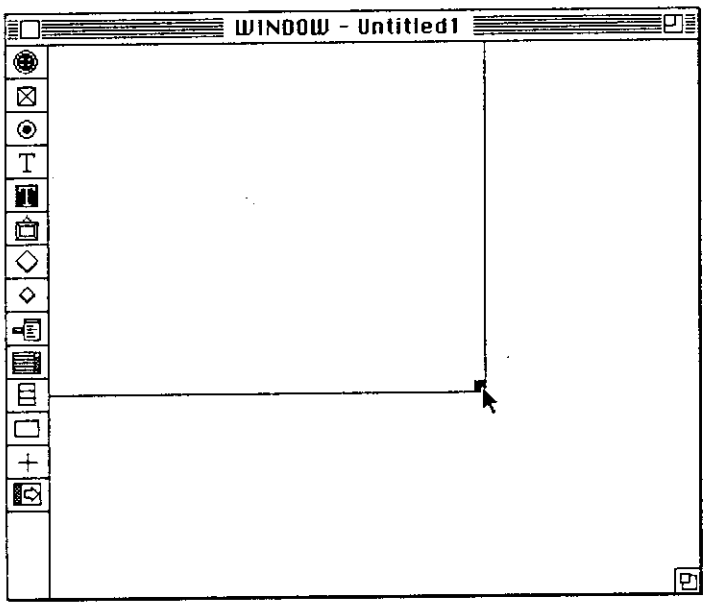


Resizing a Window

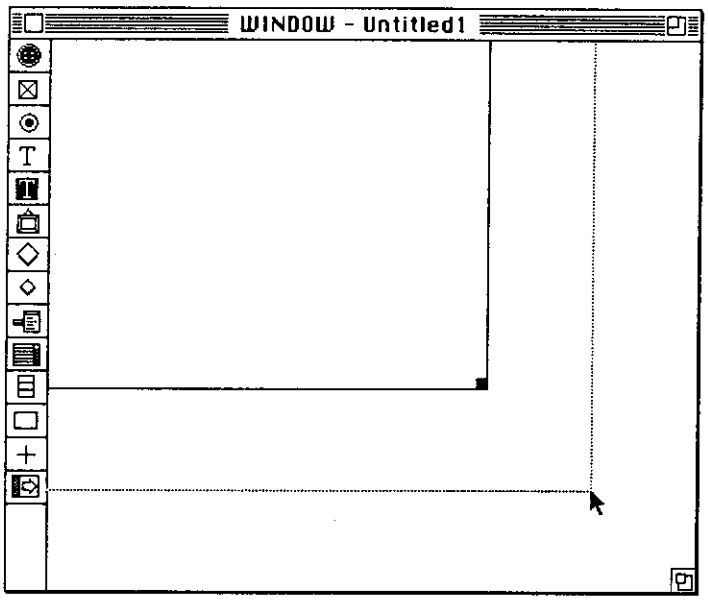
Each window created in the prototype can be resized from within the Worksheet.

To resize a window:

1. **Press the handle in the lower right hand corner with the mouse.**



2. **Drag the outline of the window to the desired size.**



3. **Release the mouse button.**

In addition, you can resize the window with the Set Location dialog box. This is done either by typing pixel coordinates, or with the resize box in the miniature screen in the Set Location dialog box.

Specifying Additional Window Attributes

Prototyper lets you specify additional window attributes, such as whether the window title bar contains a close box, or if the window should be opened when the prototype is simulated. You may also change a prototype window to a different window type, even turning a window into a dialog box or alert, or vice versa. Attributes are specified for the window in the active Worksheet. Windows, dialog boxes and alerts have different collections of attributes.

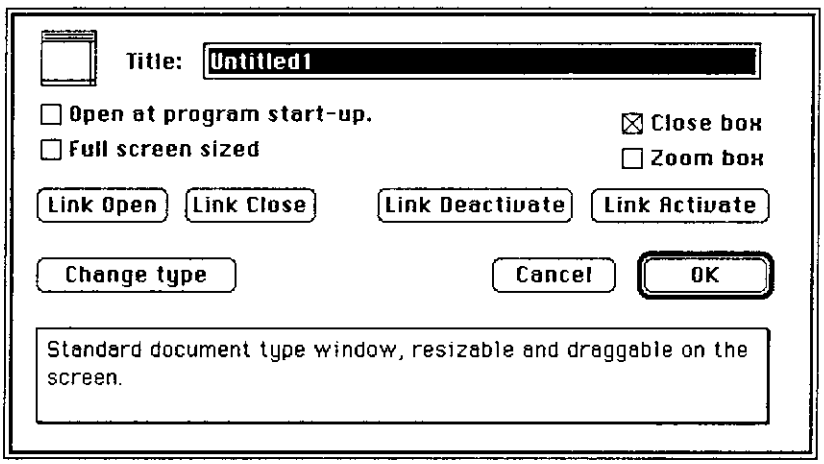
Window Info

Prototyper allows you to edit a window's title, change its type, set a window to open when the prototype is run, set a window to open to full screen size, and add a close box to a window.

You can also use opening and closing of the window to trigger various other actions in your prototype through linking, and you can even link activation and deactivation of the window in the same way. See the chapter Linking for more information.

To specify additional attributes for Windows:

1. **Make the Window's worksheet frontmost.**
2. **Choose Window Info from the Info Menu.**



3. **To change the title of the window: Enter a new title.**
4. **To designate the window to be open at the start of the prototype: Click the check box for Open at program start-up.**

5. **To give the window a close box: Click the check box for Close box.**
6. **To designate a full screen window: Click the check box for Full screen sized.**
7. **To give the window a zoom box: Click the check box for Zoom box.**
8. **To change the window type: Click the Change Type button.**

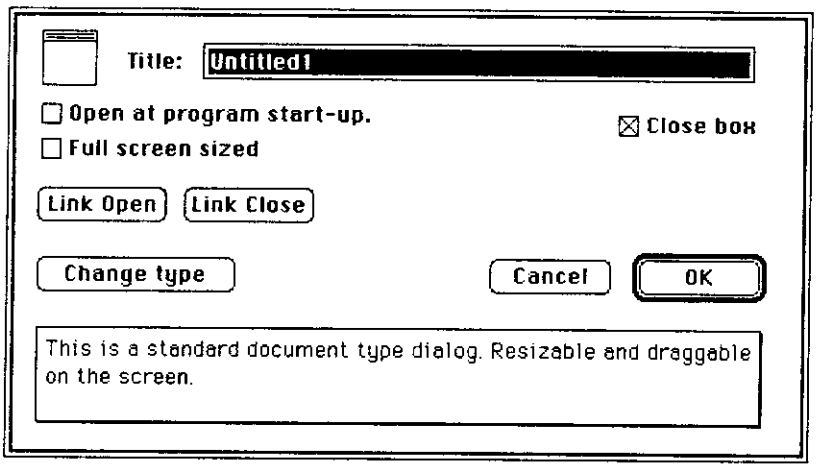
   Note: When changing a standard window to a dialog box or alert, not all the zones are transferred. Prototyper moves only transferable zones, after alerting you to the loss of the others. See Changing Window Type.
9. **Click OK or press Return.**

Dialog Boxes

You can edit a dialog box's title, change its type, give a modeless dialog a close box, or select a dialog box to open when your prototype is run. Prototyper 3.0 supports additional linking choices for modeless dialogs, described more fully in the Linking chapter.

To specify additional attributes for Dialog Boxes:

1. **Select Window Info from the Info Menu.**



2. **To change the title of the dialog box, enter a new title.**
3. **To designate a dialog box to open when the prototype is started, click the check box for Open at program start-up.**
4. **To change the type, click the Change Type button.**

Note: When changing a dialog box to an alert, not all zones are transferred.. When changing a dialog box to a window, all zones are transferred. Prototyper moves only transferable zones, after alerting you to the loss of the others. See Changing Window Type.

5. **To give a modeless dialog a close box, click the check box for close box.**
6. **Click OK or press Return.**

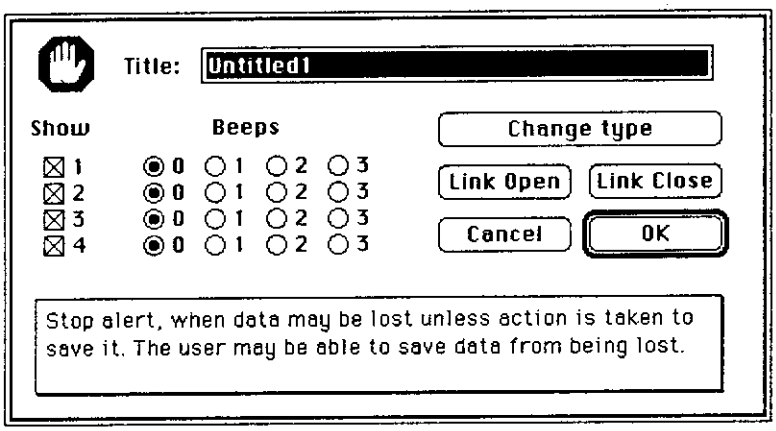
Alerts

You can edit an alert's title, or change its type. You can also specify how an alert should signal the user through each of four stages: with one or more beeps, by appearing on the screen, or by any combination of these actions.

Note: The stages of showing or beeping an alert, and setting its default button, are supported in generated code only; they do not operate during simulation.

To specify additional attributes for Alerts:

1. **Select Window Info from the Info Menu.**



2. **To change the title of the alert: Enter a new title.**

   Alerts warn of an unusual situation. This is conventionally done by sounding an initial beep, and optionally by also displaying the alert box itself. You can set four different stages for showing the alert or just beeping. A typical sequence might cause the first two occurrences of a mistake to simply beep, with subsequent occurrences displaying an alert box to explain the error.

The Show column designates whether the alert is displayed during each of the four stages. The Beeps column designates the number of times to beep during each of the four stages.

3. **To show the alert at each stage, click the appropriate check boxes to designate when the alert is displayed.**
4. **To make the alert cause a beep at each stage, click the button for the number of beeps to sound.**
5. **To change the window type, click the Change Type button.**

   Note: When changing an alert to a dialog box or window, all zones are transfered. See Changing Window Type.

6. **Click OK or press Return.**

Demo Windows

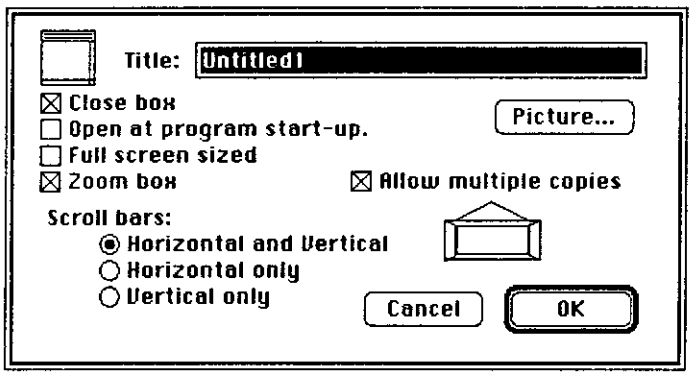
Prototyper's demonstration windows, or demo windows, allow you to integrate a fully-scrolling simulated document window into your prototype without any programming.

In simulation a demo window behaves as if it were a real operating window which could be edited by a user. You designate the contents of these demo windows by either designing and loading a picture with the picture editor, or by simply selecting a text file.

However, please note that demo windows cannot be changed to a different type of standard Macintosh window, and are not supported by Prototyper's Code Generators.

To specify additional attributes for Picture Demo Windows:

1. **Select Window Info from the Info Menu.**



2. **To change the title of the demo window: Enter a new title.**

3. **To place the picture in the demo window, click the Picture button.**

   The Picture button opens the Picture Editor. A miniature illustration of the placed picture is displayed to the lower left of the Picture button. Information about using the Picture Editor can be found in the Creating Zones chapter.

4. **To designate a demo window to be open at the start of the prototype: Click the check box for Open at program start-up.**

5. **To designate that a demo window be full screen sized: Click the check box for Full Screen Size.**

6. **To give a demo window a close box: Click the check box for Close box.**

7. **To give a demo window a zoom box: Click the check box for Zoom box.**

8. **To designate horizontal and vertical scroll bars for a demo window, select the appropriate radio button.**

   A picture demo window can scroll the contents of the window horizontally, vertically, or in both directions.

9. **Click OK or press Return.**

To specify additional attributes for Text Demo Windows:

1. **Select Window Info from the Info Menu.**

2. **To change the title of the demo window: Enter a new title.**

3. **To set the maximum horizontal character width of the text file: Enter the desired number.**

   The default width is 200 characters.

4. **To place text in the demo window, click the text button.**

   The Text File button opens a standard list dialog box. Search through your hard disk or file directory and select your text file.

5. **Click the Open button or double-click the text file.**

   The first dozen characters of the text file chosen are displayed in the dialog box to the lower left of the Text File button.

6. **To designate a demo window to be open at the start of the prototype: Click the check box for Open at program start-up.**

7. **To designate a demo window to be full screen sized: Click the check box for Open at program start-up.**
8. **To give a demo window a close box: Click the check box for close box.**
9. **To give a demo window a zoom box: Click the check box for zoom box.**
10. **To designate horizontal and vertical scroll bars for a demo window, select the appropriate radio button.**

    A text demo window can scroll the contents of the window horizontally, vertically, or both horizontally and vertically.

11. **Click OK or press Return.**

Changing Window Type

With Prototyper, you can change a window's type at any point in the design process. Any window, dialog box, or alert can be changed to any other window, dialog box, or alert.

An exception to this rule is that you cannot change Demo windows to any other type of window, nor can any type of window be changed to a Demo window.

To change a window's type:

1. **Choose Window Info from the Info Menu.**

   A Worksheet must be active on the screen to change its type.

2. **Click the Change Type button.**

   An alert box appears, warning that not all zones available in windows are transferable to dialog boxes and alerts. If you are changing from an alert or a dialog box to a window, all zones will transfer to the new window type.

3. **Click OK.**

   A dialog box appears, showing the available types of windows, dialog boxes, and alerts. The current window can be changed to any other type of window, dialog box or alert.

4. **Select the desired window icon.**

5. **Click OK or press Return.**

   If a window is changed to a dialog box or an alert, not all the zones will transfer.

Setting Window Location

Prototyper allows you to set the initial window location on a standard Macintosh monitor when the prototype is run. If a window is movable, the user of your prototype can of course move it to a different location when running the prototype.

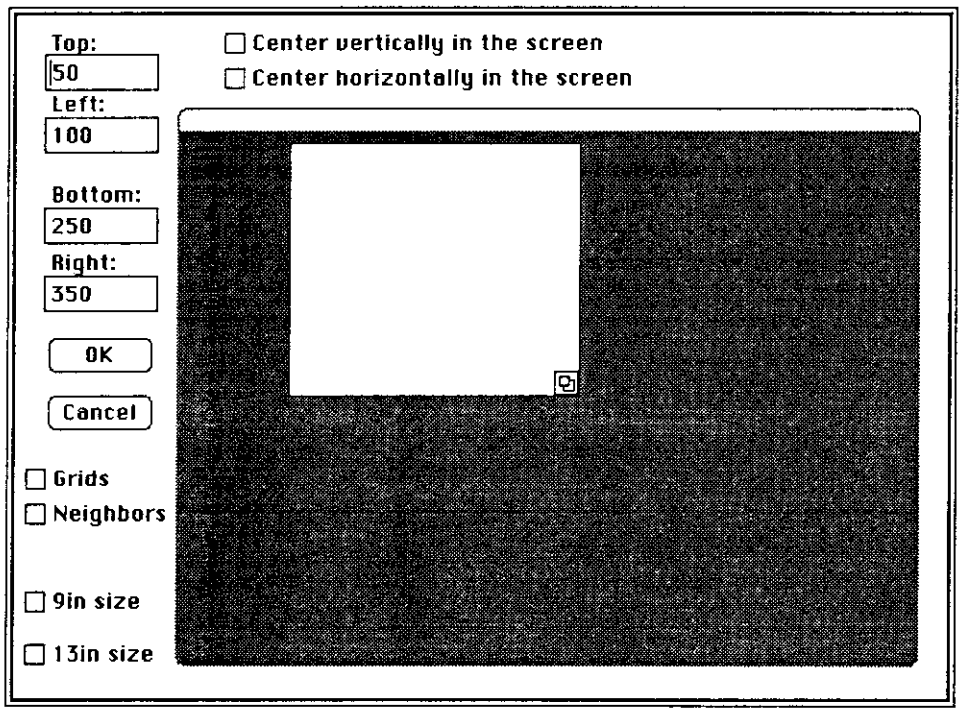
To make it easier to coordinate several windows that will share the screen together, Prototyper 3.0 lets you see the outlines of "neighboring" windows. Outlines of all prototype windows with open Worksheets are shown when you select the Show Neighbors checkbox.

The standard vertical screen position for dialog boxes is the top 1/3 of the screen. Prototyper automatically defaults all dialog boxes to the top 1/3 of the screen.

Guidelines can be displayed to show the dimensions of standard 9-inch and 13-inch Apple monitors if you are using Prototyper with a larger monitor. This helps you to set window locations more effectively for those users of your prototype who may be confined to smaller screen sizes.

To set the window's location:

1. **Open the windows worksheet and choose Set Location from the Layout menu.**



A miniature Macintosh screen is displayed, with the active window placed in default position. If you have a display larger than the standard 9" screen, the Set Location dialog box enlarges to accommodate window designs for a variety of screen sizes.

The location and size of the window can be changed in a variety of ways;

1. Drag the window to the desired location. Release the mouse button to set its location. Notice that the pixel coordinates change when you release the mouse.
2. Enter the exact pixel coordinates in the edit text areas which represent the top, left, bottom, and right corners of the window. Press the Enter key. The new window location is displayed.
3. You can also resize the window in the set location dialog by using the resize box in the bottom right corner of the window.

Note: The window size indicates the window's content area, which does not include the window's title bar.

2. **To vertically auto-center a window: Click the checkbox for Center vertically in the screen.**

   Notice that as you resize or reposition the window, the window immediately auto-centers itself on the screen. Prototyper 3.0 adjusts the position of the window, allowing for the height of the menubar, and ensures that the upper edge of the window always remains below the menubar.

3. **To horizontally auto-center a window: Click the checkbox for Center horizontally in the screen.**

4. **To use grids for alignment: Click the checkbox Grids and a 100-pixel grid appears on the miniature Macintosh screen.**

   Grid lines often make it easier to align windows that are not auto-centered but require consistent positioning with other windows in your prototype.

5. **To show the outlines of other windows in open Worksheets: Click the checkbox for Show Neighbors.**

   The outlines of all other windows in your prototype for which you have currently open Worksheets appear in the Set Location display. This helps position several windows which may appear simultaneously for maximum effectiveness.

6. **Click the OK button.**

Removing Windows

Remove windows from your prototype with the Remove Window command.

To remove a window:

1. **Open the window's Worksheet, or bring the desired Worksheet to the front by selecting it or double-clicking its title in the prototype window.**
2. **Choose Remove Window from the Special menu.**

   Choosing Remove Window will completely eliminate the window from your prototype, along with its zones. An alert will be displayed asking if you are sure you want to remove the window.
3. **Click the Cancel button or press return.**

   Note: Removing a window cannot be undone!

Duplicating Windows

Any window, dialog, or alert, along with all its zones, can be completely duplicated.

To duplicate a window:

1. **Open the window's Worksheet, or bring the desired Worksheet to the front by selecting it or double-clicking its title in the prototype window.**
2. **Choose Duplicate Window from the Special menu.**

   An exact copy of the window and zones is created. The copied window becomes the open Worksheet.

   Note: Duplicating a window will only duplicate the window and zones. Any linking created in the original window is not duplicated.

Exporting and Importing Windows

Any window, dialog, or alert can be exported from your prototype and saved into its own window document. You can develop a library of commonly used windows, or transfer a window from one prototype to another with this feature.

To export a window from your prototype:

1. **Open the window's Worksheet.**
2. **Choose Export Window from the Special Menu.**

   A standard directory dialog box appears, displaying all of the applications and files on your disk. Determine where in your hard disk or file directory you wish to save the exported window.
3. **Rename the window.**

   The window is given the same name as the prototype by default, so you may want to rename it so that it is more easily distinguished.
4. **Click the Save button.**

To import a window from another prototype:

1. **Choose Import Window from the Special Menu.**

   A standard list dialog box appears containing all of the applications and files on your disk. Find the folder or place on your disk where you keep your exported windows.

   Note: You can only import windows previously exported by Prototyper 3.0. You cannot import windows from another prototype document directly.
2. **Select the desired window.**

   The window retains the name it was given in the original prototype. If this name is the same as a window in your current prototype, you will be warned before you import the window.
3. **Click the Open button.**

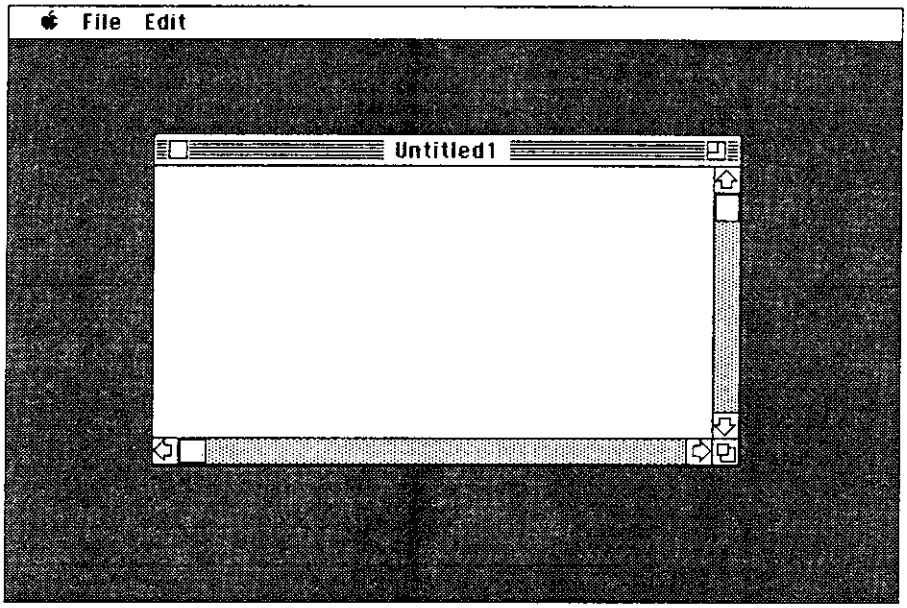
   The imported window is displayed in the prototype window.

Simulating Windows

Any window, dialog, alert, or demo window, along with all its zones, can be completely simulated within Prototyper. Although this is a simulation, there is no visual indication that the prototype is not itself a real application, making simulation very useful for demonstrations or brainstorming.

To simulate a window:

1. **Open the window's Worksheet, or bring the desired Worksheet to the front by selecting it or double-clicking its title in the prototype window.**
2. **Choose QuickLook from the Special menu.**



The window runs as if it was a full stand-alone application, complete with the currently designed menus included in the menu bar. The window is completely functional, responding as you would expect it to.

Note: If you are simulating a dialog or alert that does not have buttons to make itself go away (thereby locking you inside the dialog), you can press control-period (.) to cause the dialog to go away, exiting the simulation.

3. **To exit the simulation: Choose Quit from the simulator's File menu.**

Simulating entire Prototypes

If you want to run the entire prototype (e.g.. if your window is not set to open at program start-up), or you want to cause your window to appear on the screen as a result of some other action, choose Run from the Special menu.

Editing Existing Windows

If you need to edit or change a window that you previously created, double-click the title of the window you want to modify in the Prototype Window, or select the window and choose Open from the File menu. The window appears on the screen and is now available for modification or editing. Change the window size, location, type, etc.

Creating Zones

3. Creating Zones

Manipulating Zones 3-2
Grids 3-2
Creating Zones 3-3
Zone Coordinates 3-4
Resizing Zones 3-5
Moving Zones 3-6
Aligning Zones 3-8
Locking Zones 3-9
Description of Zone Types 3-10
- Buttons 3-10
- Icons 3-12
- Small Icons 3-15
- Check Boxes 3-18
- Radio Buttons 3-20
- Scroll Bars 3-22
- Pop-up Menus 3-24
- Static Text Zones 3-26
- Edit Text Zones 3-27
- Manipulating text for Static or Edit Text Zones 3-28
- Lists 3-29
- Rectangles 3-31
- Lines 3-33
- Pictures 3-34
- Plugin Buttons 3-37
- Plugin Toggles 3-38
- Plugin Gauges 3-39
- Palettes 3-40
- HotSpots 3-42

Creating Zones

Windows contain graphical objects such as icons and pictures, text areas, and components such as buttons, scroll bars, Check Boxes, and Pop-up menus. Some objects such as radio buttons, Check Boxes, and Pop-up menus are used to change settings or to modify future actions.

For simplicity, this manual refers to everything (graphical objects, text areas, and components) that can be placed inside a window as a Zone.

Prototyper 3.0 contains fifteen different types of zones, and you can add zones in three basic types. They are:

Button
Check Box
Radio Button
Static Text
Edit Text
Picture
Icon
Small Icon
Pop-up Menu
List
Palette
HotSpot - Rectangle
Line
Scroll Bar
Plugin Button *(if installed with Plugin Installer)*
Plugin Toggle *(if installed with Plugin Installer)*
Plugin Gauge *(if installed with Plugin Installer)*

Standard document windows usually comprise only edit text and scroll bars. Most of the other zones find more frequent use in Dialog and Alert Boxes.

This chapter describes how to create, resize, move and specify additional attributes for zones. You will also learn how to use the Icon Editor to create an icon, and the Picture Editor to import a picture.

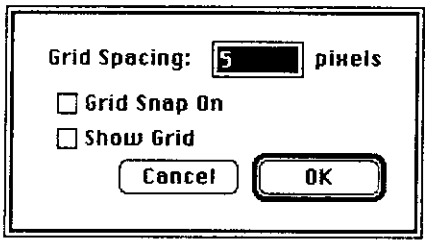
Manipulating Zones

Grids

When creating zones in a window, you will need to align them accurately. To help line them up precisely, Prototyper 3.0 provides alignment grids. Grids can control the positioning of zones with the Grid Snap feature, and can either be displayed in the Worksheet or remain invisible.

To use the Grids command:

1. **Choose Grids from the Layout menu.**



   The grid dialog appears.

2. **Set the grid spacing in pixels by entering in the desired number.**
3. **Click the Grid Snap On Check Box.**

   Checking the Grid Snap On Check Box causes zones to snap to the nearest grid position when they are created, moved, resized, etc.

4. **Click the Show Grid Check Box.**

   Checking the Show Grid Check Box displays the grid in the Worksheet. This choice can be used with or without the Grid Snap feature, to help you align your zones in windows.

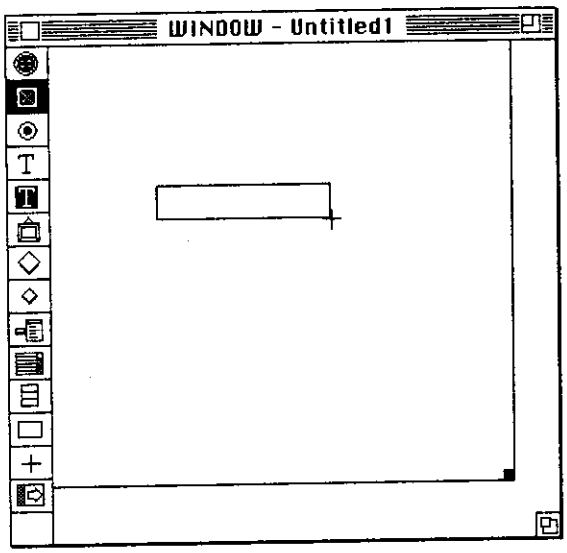
5. **Click OK.**

Creating Zones

Zones set up conditions for future actions, or cause immediate actions. Either way, they are a directed method for facilitating user interaction.

To create zones (of any type) in the Worksheet:

1. **Click on a zone tool in the palette at the left side of the Worksheet.**



   The pointer changes to a crosshair and the tool is highlighted. A Checkbox zone is being created in this illustration; however, the process is analogous for all zones.

2. **Move the crosshair to the position in the window marking the upper left corner of the zone.**

3. **Press and draw a rectangle to indicate the size of the zone, moving towards the lower right corner.**

   Note:Simply clicking in the worksheet without dragging will drop the zone in at it's 'default' size.

4. **Release the mouse.**

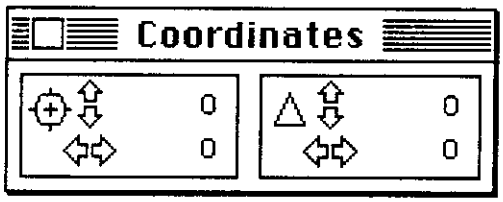
   Zones respond to the standard Cut, Copy, Paste, and Duplicate commands. They can be Cut, Copied and Pasted into the same window or another window. Zones can be removed from the Worksheet by choosing Cut, or by pressing the delete or backspace key.

Zone Coordinates

The Coordinates window lets you control an individual zone's size and position on the screen with to-the-pixel accuracy. This helps you to keep various zones consistent throughout an application by verifying their size and position.

To enable the coordinates window:

1. **Select Coordinates from the Layout menu.**



The left side of the window displays the pixel location of the top left part of the selected zone, with 0,0 being the upper left most part of the window Worksheet.

The right side of the window displays the pixel dimensions of the selected zone. The top number displays the vertical size and the bottom number the horizontal size of the selected zone.

2. **Select a zone to see its size and location in the window.**

As you move or resize zones, the results can be monitored with the Coordinates window. See the next sections n resizing and moving zones.

Note: The Coordinates window is not a floating window, so make sure when you select a zone that your Worksheet does not cover the coordinate display.

3. **To close the Coordinates window: Click the close box.**

Note: Pressing both the Option and Command keys causes Prototyper 3.0 to display the borders of all zones in the active window. When these keys are released, the borders disappear. This is particularly helpful when making sure zones do not overlap.

Resizing Zones

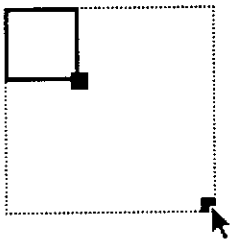
The size of a zone in a window can be made larger or smaller. For certain zones, such as buttons, pictures, and rectangles, the zone itself is made larger or smaller, while other zones such as radio buttons and Check Boxes allow the zone area to be made larger or smaller to display a title or text. A radio button or Check Box always defaults to the standard Macintosh size.

To resize a zone:

1. **Select the zone in the Worksheet.**

   A handle appears on the bottom right hand corner.

2. **Drag the cursor to change the size of the zone.**



   Press the Option key while resizing to maintain the horizontal width. Press the Command key while resizing to maintain the vertical width.

3. **Release the mouse button when the desired size is reached.**

   As mentioned in a previous section on using the coordinates window, you may want to make precision zone resizing with to-the-pixel accuracy. This is possible using the arrow keys on your keyboard and nudging the zones size any direction one pixel at a time.

4. **To nudge the size of a zone: Hold down the command key and press the appropriate arrow key.**

   The *left* arrow key causes the zone to decrease in size horizontally by one pixel.

   The *right* arrow key causes the zone to increase in size horizontally by one pixel.

   The *down* arrow key causes the zone to increase in size vertically by one pixel.

   The *up* arrow key causes the zone to decrease in size vertically by one pixel.

   Holding down the command and arrow key causes the zone to continue to change its size. The Coordinates window displays this resizing interactively.

Moving Zones

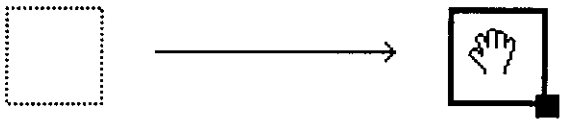
Zones can be moved to different locations in a window. Prototyper 3.0 provides two different displays for moving a zone; you can move the zone itself, or an outline of the zone can be moved. Set a preference for moving zones by choosing Preferences from the Info menu. Click your desired preference.

To move a zone:

1. **Select the Zone in the Worksheet.**

   The pointer changes to a grabber hand for repositioning the zone. If the "drag outline" preference has been chosen, an outline of the zone is created. The zone retains its original position until the mouse button is released.

2. **Drag the zone or zone outline to the new location.**



3. **Release the mouse when the desired location is reached.**

   As mentioned above, precision zone resizing with to-the-pixel accuracy is possible by using the arrow keys on your keyboard and nudging the zones size any direction one pixel at a time. Using the Coordinates window gives you quantitative feedback as you change the size.

4. **To nudge the location of a zone: Press the appropriate arrow key.**

   The *left* arrow key causes the zone to move left one pixel.

   The *right* arrow key causes the zone to move right one pixel.

   The *down* arrow key causes the zone to move down one pixel.

   The *up* arrow key causes the zone to move up one pixel.

   Holding down the arrow key causes the zone to continue to move. The Coordinates window displays the movement interactively.

5. **To move more than one zone, select the desired zones.**

   There are a variety of ways to select multiple zones; you can:

   1. Select each zone by holding down the Shift key and pressing anywhere in the zone. Repeat this procedure until all desired zones are selected.
   2. If you wish to select all zones: Choose Select All from the Edit menu.
   3. Use the cursor and drag a rectangle around those zones you wish to select. This is done in the same manner that you would select files or folders in the Finder.

6. **Once all desired zones are selected, move them as described above: by dragging with the mouse, or by nudging with the arrow keys.**

Aligning Zones

Zones can be aligned automatically as a group. This function can apply to specific zones within a single window, or to all of the zones in a window. Zones can be aligned left, right, top, bottom, and center. By using this feature in conjunction with Prototyper's on-screen grids and the Coordinate window, you can accurately align your zones.

Note: Zones are aligned by their Worksheet boundaries, rather than the visible characteristics of the zone. For example, you might design an icon that has only a 10 pixel width and 10 pixel height; when aligned, it is repositioned using the original 32 by 32 pixel size of the icon zone as a guide.

To align zones:

---

1. **Select the zones you wish to align.**

   There are a variety of ways to select multiple zones; you can:

   1. Select each zone by holding down the Shift key and pressing anywhere in the zone. Repeat this procedure until all desired zones are selected.
   2. If you wish to select all zones: Choose Select All from the Edit menu.
   3. Use the cursor and drag a rectangle around those zones you wish to select. This is done in the same manner that you would select files or folders in the Finder.

2. **Choose one of the Align options from the Layout Menu.**

   You may align zones: Top, Bottom, Center, Left or Right.

   Note: Pressing both the Option and Command keys causes Prototyper 3.0 to display the borders of all zones in the active window. When these keys are released, the borders disappear. This is particularly helpful when making sure zones do not overlap.

Locking Zones

Zones can be locked down when you have placed them in their final positions, so that you do not accidentally move or alter them later as you are editing your window design.

Locked zones can still be resized or deleted.

To lock a zone:

1. **Select the zones you wish to lock.**
2. **Choose Lock zone from the Layout menu.**

   The selected zone is now locked. If you want to nudge the zone a few pixels, without unlocking the zone, you may do so with the arrow keys.

3. **To unlock the zone: Choose Unlock zone from the Layout menu.**

   Note: Pressing both the Option and Command keys causes Prototyper 3.0 to display the borders of all zones in the active window. When these keys are released, the borders disappear. This is particularly helpful when making sure zones do not overlap.

Description of Zone Types;

Zones, like windows, have additional attributes specific to each zone type. They can be dimmed, or initially selected when the prototype is run. To set these attributes, double-click the selected zone, or select the zone and choose Zone Info from the Special menu.

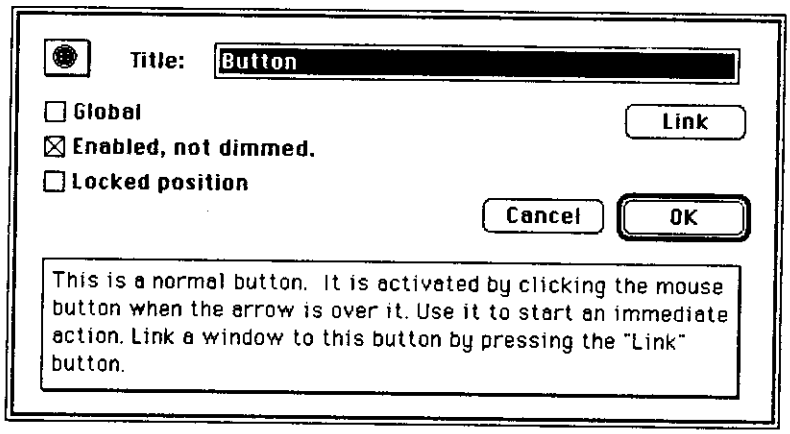
Buttons

A push button is a screen object labeled with text, which when clicked or pressed performs an instantaneous action described by its label, such as "Cancel" or "Quit". If one button is the default, (pressing Return or Enter has the same result as pressing the button) then it is outlined with a heavy black border.

Prototyper 3.0 allows buttons in windows be linked to open other windows, dialogs or alerts, or be enabled instead of dimmed when the prototype is run. Buttons in dialogs can be enabled, close the dialog, open an alert, or be specified as the default button. Buttons in alerts can be specified as the default, be enabled or close the alert. The dialog box for setting these attributes contains only those attributes appropriate to the button's context in a window, dialog, or alert.

To set additional attributes for a button zone:

1. **Double-click the button zone in the Worksheet.**



   Or, you can select the zone, then choose Zone Info from the Info menu.

2. **To enable the button: Click the Enabled Check Box.**

   When a button is enabled, it can be clicked when the prototype is run; otherwise, it is dimmed.

3. **To have the button close the dialog: Click the Check Box for Close dialog when checked.**

4. **To set the default button: Click the Check Box for Default.**

   Only one button in a dialog and alert can be specified as the default. If one button is already set as the default button, setting another button to be the default automatically unchecks the previously defaulted button. Prototyper 3.0 warns you that a default button has already been set.

5. **To have the button 'remember' it's enable status: Click the Global Check Box.**

   A global button will have it's enabled/disabled state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

6. **To link a button to an action in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in the later chapter called "Linking". In summary, you can link buttons to:

   - Open a window, dialog, alert, or demo window.
   - Close a window or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.

   Note: Buttons in windows can be linked to open windows, dialogs and alerts. Buttons in dialogs can be linked to open alerts. Buttons in alerts cannot be linked; they may only close the alert.

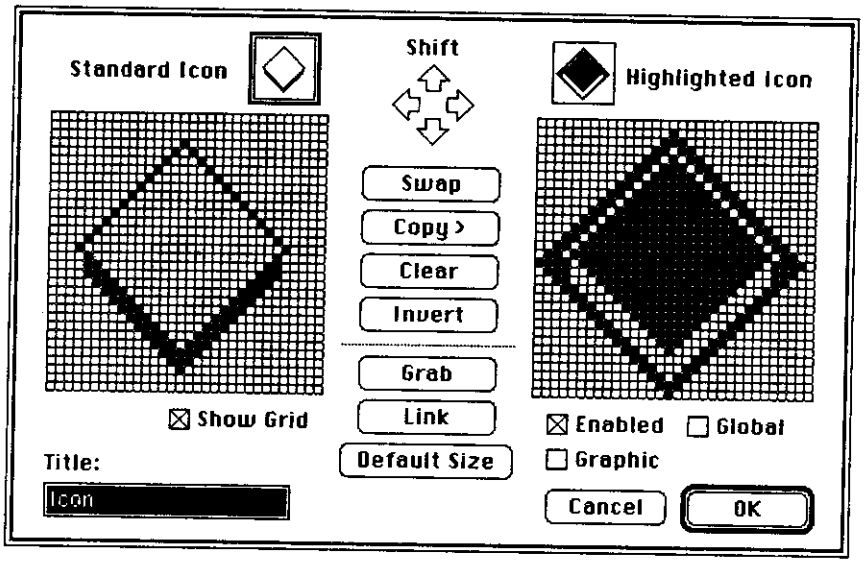
7. **Click OK or press Return.**

Icons

An icon is a small picture, composed of a bitmap of 32 by 32 pixels. Icons are a powerful way to communicate an idea, action or attribute without relying on the reading of text by the user. With Prototyper, you can use icons just like buttons, to perform an instantaneous action symbolized by the icon's appearance.

To create an icon:

1. **Double-click the icon zone in the Worksheet.**



This displays the Icon Editor.

The Icon Editor lets you create and edit icons. The icon on the left represents the standard form of the icon. The icon to the right represents the highlighted form of the icon — what it looks like when it is selected.

Either form can be worked on by selecting it at the top of the Icon Editor. When the Icon Editor is opened, the standard icon is initially selected. All buttons, check boxes, and arrows operate on the icon that is selected at the top of the Editor.

2. **Enter the title for the icon.**

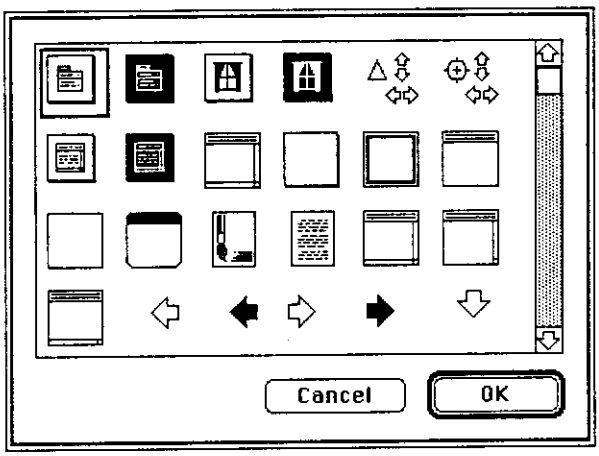
The title is used to identify this particular zone in generated code.

3. **Design the icon.**

Icons can be designed in three ways:

1. Click the Clear button. This completely clears the icon that is selected. Use the mouse to design a new icon.
2. Use the mouse to clear away some or all of the current icon design. Finish the design.
3. Use the Icon Grabber to select an icon from the group stored in the prototype. As each new icon is created, it is added to this group. An unlimited number of icons can be stored in the Grabber.

   a. Click the Grab button.



   This displays the Icon Grabber.

   b. Scroll through the icons to the desired icon.

   c. Click OK.

4. **Create a highlighted icon.**

To create a highlighted version of the current icon: Click the Invert button.

5. **Swap Icons**

To swap the standard icon and highlighted icon: Click the Swap button.

6. **Shift Icon Locations**

To shift the icon up, down, right, or left:Click the desired direction arrow.

One click on any of the direction arrows moves the icon one row in the selected direction. The icon "wraps" around; pixels moving off to the left immediately reappear from the right.

7. **To show the Grids: Click the show grids Check Box.**

   The grids are displayed when the Check Box is checked, otherwise they are hidden.

8. **Set the Icon Size Default**

   The standard size for an icon is 32 pixels by 32. The icon editor is set up to create a standard size icon. However, you can change the icon's size to be larger or smaller than the standard size — this doesn't alter the number of pixels in the icon's design, but causes it to be expanded or compressed when it is displayed. If you change the size of the icon in this way, it will appear distorted in the Worksheet.

   To return the icon to the standard default size: Click the Default button

9. **To enable the Icon: Click the Enabled Check Box.**

   When an icon is enabled, during simulation and in generated code the icon can be clicked by the user and will display its highlighted version. If the icon is not enabled, it will be grayed out and any links it has will be disabled also.

10. **To inactivate the Icon: Click the Graphic Check Box.**

    When an icon's graphic setting is checked, during simulation the icon's links will be disabled.

11. **To have the icon 'remember' it's enable status: Click the Global Check Box.**

    A global icon will have it's enable/disable state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

12. **To link an icon to evoke an action in your prototype: Click the Link button.**

    This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities is described in its own chapter called "Linking". In summary, you can link icons to:

    - Open a window, dialog, alert, or demo window.
    - Close a window or demo window.
    - Disable and Enable menu titles.
    - Disable and Enable individual menu commands.
    - Disable and Enable Zones and Custom Controls.
    - Open Page Set-up and Printing Dialogs.
    - Open File and Save Dialogs.
    - Check or Toggle menu commands.
    - Select Radio Buttons.
    - Play Sounds.

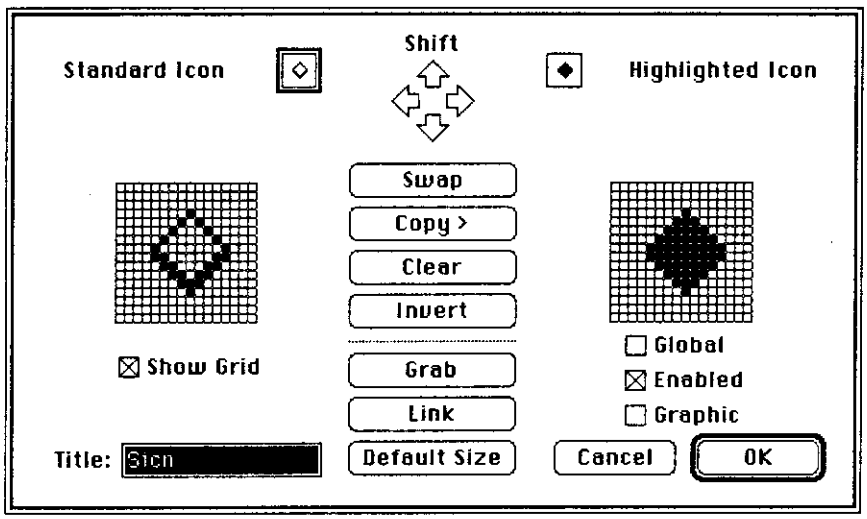
12. **To exit the Icon Editor: Click OK or press Return.**

Small Icons

A small icon is a small picture, composed of a bitmap of 16 by 16 pixels. Small Icons are a powerful way to communicate an idea, action or attribute without relying on the reading of text by the user. With Prototyper, you can use small icons just like buttons, to perform an instantaneous action symbolized by the small icon's appearance.

To create a small icon:

1. **Double-click the small icon zone in the Worksheet.**



This displays the Small Icon Editor.

The Small Icon Editor lets you create and edit small icons. The small icon on the left represents the standard form of the small icon. The small icon to the right represents the highlighted form of the small icon — what it looks like when it is selected.

Either form can be worked on by selecting it at the top of the Small Icon Editor. When the Small Icon Editor is opened, the standard small icon is initially selected. All buttons, check boxes, and arrows operate on the small icon that is selected at the top of the Editor.

2. **Enter the title for the small icon.**

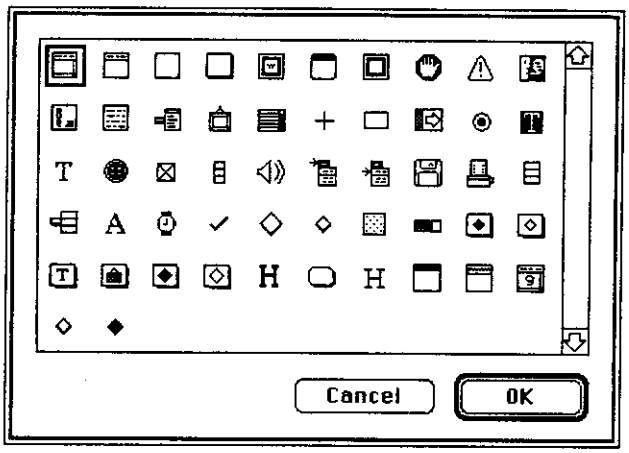
The title is used to identify this particular zone in generated code.

3. **Design the small icon.**

Small Icons can be designed in three ways:

1. Click the Clear button. This completely clears the small icon that is selected. Use the mouse to design a new small icon.
2. Use the mouse to clear away some or all of the current small icon design. Finish the design.
3. Use the Small Icon Grabber to select a small icon from the group stored in the prototype. As each new small icon is created, it is added to this group. An unlimited number of small icons can be stored in the Grabber.

   a. Click the Grab button.



   This displays the Small Icon Grabber.

   b. Scroll through the small icons to the desired small icon.

   c. Click OK.

4. **Create a highlighted small icon.**

To create a highlighted version of the current small icon: Click the Invert button.

5. **Swap Small Icons**

To swap the standard small icon and highlighted small icon: Click the Swap button.

6. **Shift Small Icon Locations**

To shift the small icon up, down, right, or left: Click the desired direction arrow.

One click on any of the direction arrows moves the small icon one row in the selected direction. The small icon "wraps" around; pixels moving off to the left immediately reappear from the right.

7. **To show the Grids: Click the show grids Check Box.**

   The grids are displayed when the Check Box is checked, otherwise they are hidden.

8. **Set the Small Icon Size Default**

   The standard size for a small icon is 16 pixels by 16. The small icon editor is set up to create a standard size small icon. However, you can change the small icon's size to be larger or smaller than the standard size — this doesn't alter the number of pixels in the small icon's design, but causes it to be expanded or compressed when it is displayed. If you change the size of the small icon in this way, it will appear distorted in the Worksheet.

   To return the small icon to the standard default size: Click the Default button

9. **To enable the Small Icon: Click the Enabled Check Box.**

   When a small icon is enabled, during simulation and in generated code the small icon can be clicked by the user and will display its highlighted version. If the small icon is not enabled, it will be grayed out and any links it has will be disabled also.

9. **To inactivate the Small Icon: Click the Graphic Check Box.**

   When a small icon's graphic setting is checked, during simulation the small icon's links will be disabled.

10. **To have the small icon 'remember' it's enable status: Click the Global Check Box.**

    A global small icon will have it's enable/disable state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

11. **To link a small icon to evoke an action in your prototype: Click the Link button.**

    This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities is described in its own chapter called "Linking". In summary, you can link small icons to:
    - Open a window, dialog, alert, or demo window.
    - Close a window or demo window.
    - Disable and Enable menu titles.
    - Disable and Enable individual menu commands.
    - Disable and Enable Zones and Custom Controls.
    - Open Page Set-up and Printing Dialogs.
    - Open File and Save Dialogs.
    - Check or Toggle menu commands.
    - Select Radio Buttons.
    - Play Sounds.

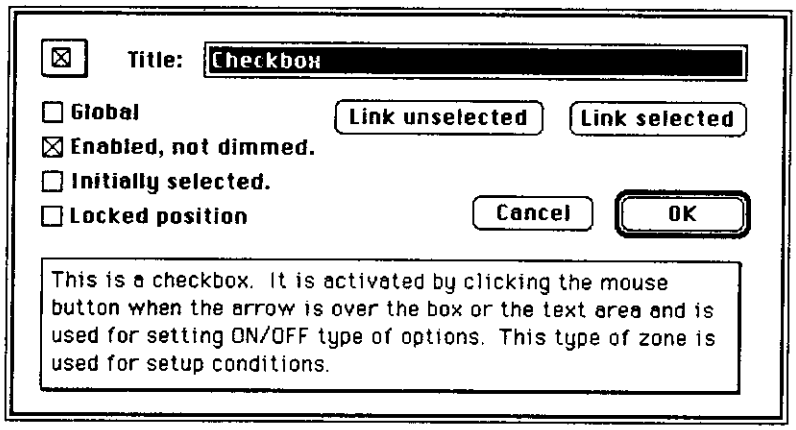
12. **To exit the Small Icon Editor: Click OK or press Return.**

Check Boxes

Check Boxes act like toggle switches. The value is "On" when a box is checked, otherwise it is off. Check Boxes should not cause any immediate action. They are used to set up conditions and let the user choose options. Check Boxes appearing together in a window work independently of one another, in contrast with the behavior of Radio Buttons (described below). Each Check Box includes a title to identify its function.

To set additional attributes for the Check Box zone:

1. **Double-click the Check Box zone in the Worksheet.**



2. **Enter the title.**

   This is the title that appears in generated source code to identify this particular zone.

3. **To enable the Check Box: Click the Enabled Check Box.**

   Enabling a Check Box means that when the prototype is run, it is not dimmed and the user can select it.

4. **To designate initially checked: Click the Initially Checked Check Box.**

   The box is initially checked when the prototype is un.

5. **To have the checkbox 'remember' it's settings: Click the Global Check Box.**

   A global checkbox will have it's check state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

6. **To link a Check Box to evoke an action: Click the Link unchecked button or the Link checked button.**

   The Link unchecked button lets you define the linking action that occurs when the Check Box is unchecked by the user. The Link checked button lets you define the linking action that occurs when the Check Box is checked by the user.

   Both of these buttons lead to the Link Dialog, described in the chapter on Linking. In summary, you can link Check Boxes to:

   - Open a window, dialog, alert, or demo window.
   - Close a window or demo window.
   - Disable and Enable menu titles.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.

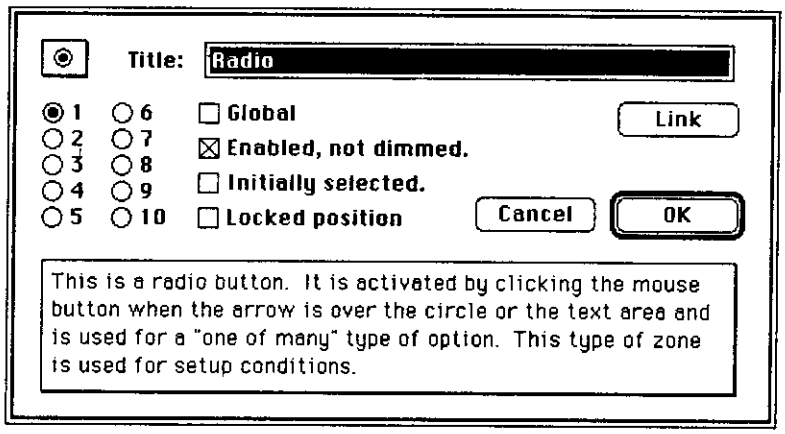
7. **Click OK or press Return.**

Radio Buttons

Radio buttons occur in groups, of which only ONE radio button may be selected. They are called radio buttons because they act like the buttons on a car radio. Radio buttons do not cause immediate action. They are used to set up mutually exclusive conditions. Radio buttons are titled to identify what each button does. Radio buttons serve a use similar to Popup Menus and can be replaced by Popup Menus, especially if the radio group has many items.

To set additional attributes for a radio button zone:

1. **Double-click the radio button zone in the Worksheet.**



2. **Enter the title.**

   The title appears in generated source code to identify this particular zone.

3. **To enable the radio button: Click the Enabled Check Box.**

   Enabling a radio button means that when the prototype is run, it is not dimmed and the user can select it.

4. **To designate as initially selected: Click the Initially Checked Check Box.**

   Selecting a radio button as initially selected causes it to be the selected radio button of the group when the prototype is run.

5. **To group radio buttons: Within the Zone Info dialog, select the group number that the zone will belong to.**

   Prototyper 3.0 lets you designate up to ten separate groups of radio buttons within each window, with an unlimited number of radio buttons per group.

By selecting group 1 for all buttons, they will work together with only one radio button selectable at a time. If two radio buttons are assigned to group 1 and two other radio buttons are assigned to group 2, each group functions independently of the other.

6. **To have the radio button 'remember' it's settings: Click the Global Check Box.**

   A global radio button will have it's selected state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

7. **To link a Radio Button to evoke an action in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in the later chapter "Linking". In summary, you can link Radio Buttons to:

   - Open a window, dialog, alert, or demo window.
   - Close a window, dialog, alert, or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.

   Note: Zones in windows & dialogs can be linked to open windows, dialogs and alerts. Zones in alerts can only be linked to close the alert.

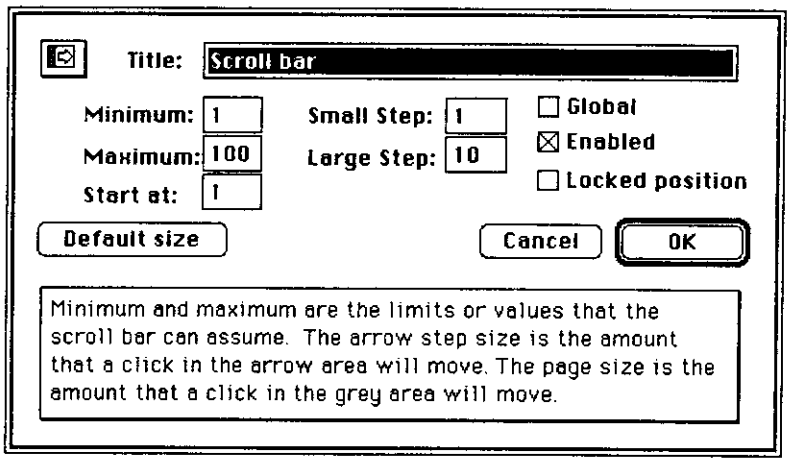
8. **Click OK or press Return.**

Scroll Bars

Scroll bars consist of a light grey rectangle with an arrow in a box at each end. Inside the scroll bar is a white box called the scroll box. You can create both vertical and horizontal scroll bars in a window. They are used to scroll the data within a window.

To set additional attributes for the scroll bar zone:

1. **Double-click the scroll bar zone in the Worksheet.**



2. **Set Minimum and Maximum Stepping.**

   Minimum and maximum stepping is adjusted by the Small Step and Large Step limits (in pixels) that the scroll bar uses when it moves the scroll box. The Small Step step is the amount that a click in the arrow area of the scroll bar moves the scroll box. The Large Step is the amount that a click in the grey area of the scroll bar moves the scroll box.

   Note: These values are used in code generation and simulation for demonstration purposes. The scroll bar values are not linked to any object and do not directly cause any action. If you use a scroll bar for a window, the window contents will not be linked to the scroll bar and it will not scroll unless you modify the generated code.

   If the scroll bars are positioned close to the edge of the window, (where scroll bars would normally appear in a scrollable document window), they will snap into place during a QuickLook or Run simulation. These scroll bars are resized with the window if it is resized using the grow box or zoom box.

3. **Set the initial value for the scroll bar.**

   The initial value of a scroll bar (or custom slider CDEF) can be set using the Start At value in pixels. This is the number of pixels from the lower limit of the scroll bar.

4. **Set Minimum and Maximum Values for the scroll bar.**

   Minimum and maximum values are adjusted by the Minimum and Maximum text fields, and limit the values that the scroll bar will assume.

5. **To set the scroll bar to its default width: Click the Default Width button.**

   If you have created scroll bars wider than 16 pixels, you can quickly resize them to the standard width without changing the size of the zone in the Worksheet.

6. **To have the scroll bare 'remember' it's settings: Click the Global Check Box.**

   A global scroll bar will have it's Value saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

7. **Click OK or press Return.**

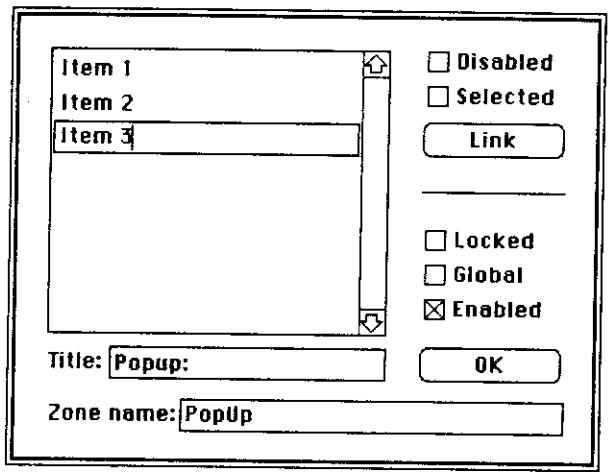
Pop-up Menus

Pop-up menus consist of a title (a static text label) followed by a colon, which in turn precedes the Pop-up menu's list of related items. This list of items is contained in a one pixel drop shadow box, with only the current selected value displayed. When the user presses on this box, the Pop-up menu appears under the pointer with the urrent value checked and highlighted. If the Pop-up menu has a title, it is also highlighted.

Pop-up menus are most commonly used to conserve screen real estate while letting the user choose from a series of values that would otherwise have to be placed in scrolling lists or radio buttons. An example might be a Pop-up menu for font, font size, and text style within a dialog.

To set additional attributes for Pop-up menu zones:

1. **Double-click the Pop-up menu zone in the Worksheet.**



2. **Enter the title.**

   The title appears in generated source code to identify this particular zone and as the Pop-up title in the window itself. Make sure that the title is followed by a colon. If you do not wish your Pop-up menu to have a title, delete all characters in this area.

3. **To enter the values for the Pop-up menu: Select the the first item in the rectangle in the list so that "Untitled" is highlighted.**

   Now you can type the value, pressing Return or Tab to advance the rectangle to the next line.

   Note: Pop-up menus cannot have key board command equivalents, unlike regular menus and hierarchical menus.

4. **To rearrange the order of Pop-up menu items:**

   1. Press on the desired item and keep the mouse depressed. The pointer changes to a hand.
   2. To place the item between two other commands, move the positioning rectangle between the two existing commands.
   3. When the desired location is reached, release the mouse button and the item snaps into place.

5. **To make a Pop-up item initially selected: Click the Selected Check Box.**

6. **To disable a Pop-up item: Click the Disabled Check Box.**

   Note: To place a separator line between menu items, simply press the dash key and click the Disabled Check Box.

7. **To have the Pop-up 'remember' it's settings: Click the Global Check Box.**

   A global Pop-up menu will have it's selected menu item saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

8. **To link a Pop-up menu item to evoke an action in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in the later chapter "Linking". In summary, you can link Pop-up menu items to:

   - Open a window, dialog, alert, or demo window.
   - Close a window, dialog, alert, or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.

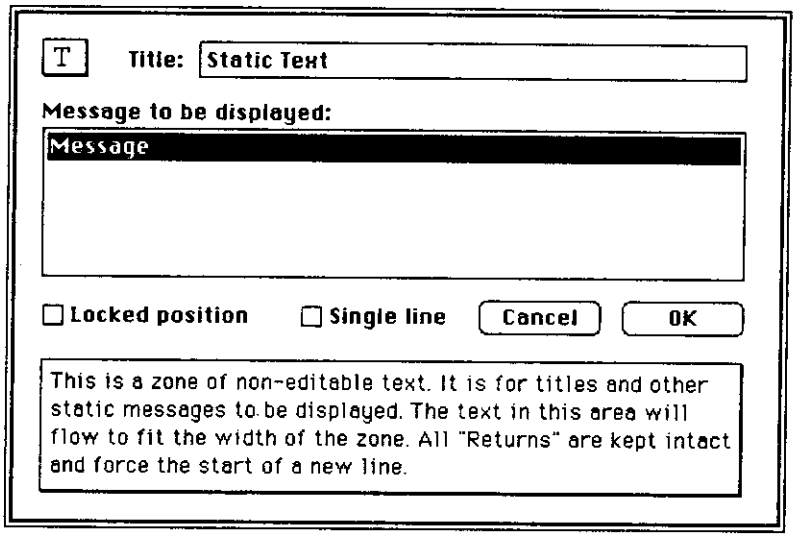
9. **To close the dialog: Click OK**

Static Text Zones

Static Text is the name given to non-editable lines of text in windows, dialogs and alerts. If text extends past the zone area in the window, the text cab either automatically flow to fit the rectangle area of the zone, or be clipped to a single line, depending on the setting of the "Single Line" Check Box. Static text is used for titles and other static messages displayed inside a window.

To set additional attributes for a static text zone:

1. **Double-click the static text zone in the Worksheet.**



2. **Enter the title.**

   This is the title that appears in generated source code to identify this particular zone.

3. **Press Tab to select the area titled "Message to be displayed."**
4. **Enter the text that will be seen in the static text zone.**

   Note: Static text is limited to 255 characters, extra text will be truncated when you close the dialog.

5. **Click OK or press Return.**

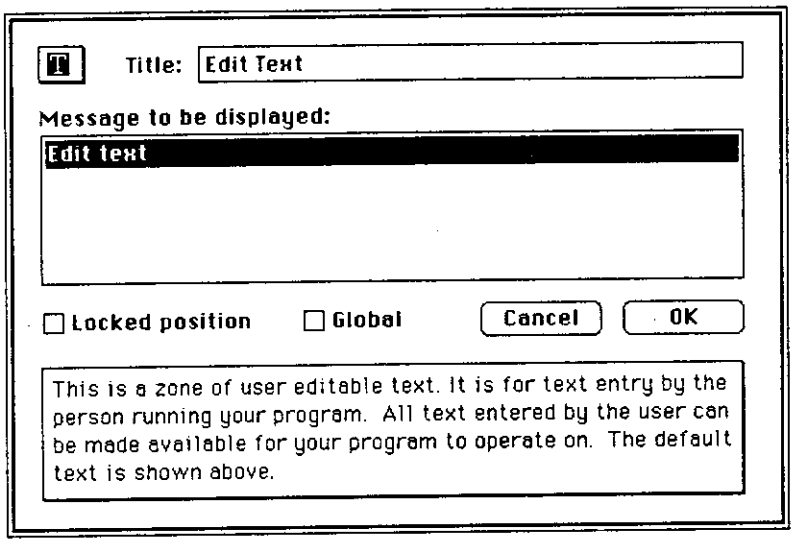
Edit Text Zones

Edit text supports text entry by the user of the prototype application. All text entered by the user is available in the generated source code.

You can use Cut and Paste in text entry areas.

To set additional attributes for an edit text zone:

1. **Double-click the edit text zone in the Worksheet.**



2. **Enter the title.**

   This title appears in generated source code to identify this particular zone.

3. **Press Tab to select the area titled "Message to be displayed."**

4. **Enter the text that will be seen in the window.**

   This is the default text that is displayed when the prototype is run. If you don't want default text to show, then this text area should be left blank.

5. **To have the text 'remember' it's contents: Click the Global Check Box.**

   A global text zone will have it's contents saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

6. **Click OK or press Return.**

Manipulating text for Static or Edit Text Zones

The font, point size, and justification can be changed for static text and edit text zones. The chosen text attributes appear in the Worksheet, the simulator, and in generated code (after compilation). When changing the font style or point size, certain guidelines must be followed.

1. Newly created text areas assume the font and style of your most recent selections from the Font and Style menus.
2. Windows: Static and edit text zones can take different font styles and point sizes.
3. Dialogs: Static text and edit text zones can be different font styles or point sizes.
4. Alerts: Text zones can be different font styles or point sizes

To change the font or point size:

---

1. **Select the static text or edit text in the Worksheet.**
2. **Choose the desired font and point size from the Font menu.**

   The font style and point size are changed immediately in the Worksheet.

To change text justification:

---

1. **Select the static text or edit text in the Worksheet.**
2. **Choose the desired justification from the Style menu.**

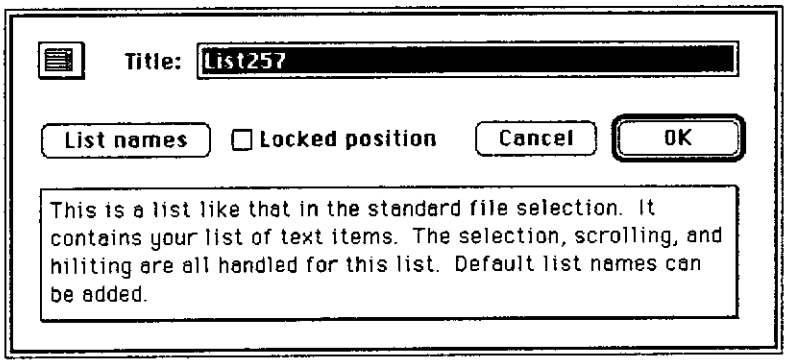
   The justification is changed immediately in the Worksheet.

Lists

Lists are used to select from a variable number of items. The selection, scrolling, and highlighting are handled by Prototyper 3.0 for lists.

To set additional attributes for the list zone:

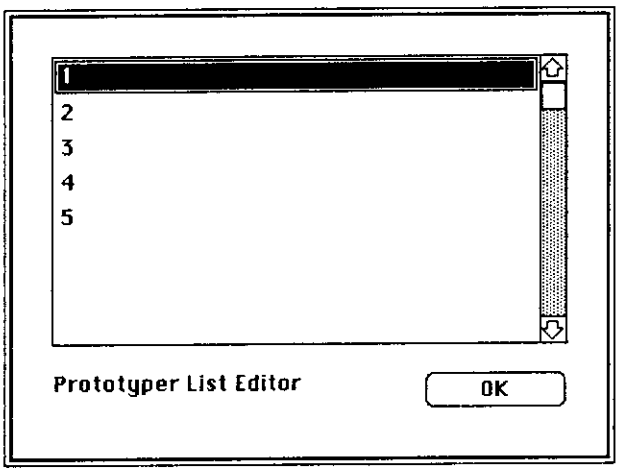
1. **Double-click the list zone in the Worksheet.**



2. **Enter the title.**

   The title appears in generated source code to identify this particular zone.

3. **To set the list contents: Click the List names button.**



Enter the default text for each cell of the list, pressing tab after each one to advance the cursor . If no default lists are set, the list displays the Prototyper 3.0 default item list when the prototype is run. To create new entries when the list is full, click on the down arrow in the scrollbar. There is no limit to the number of items in a list, but

hundreds of items could slow down the creation and operation of the list, which is created with the Macintosh List Manager.

4. **Click OK or press Return.**

To change the font or point size:

1. **Select the list zone in the Worksheet.**
2. **Choose the desired font and point size from the Font menu.**

   The font style and point size are changed immediately. This is apparent when displaying the window in simulation: in the Worksheet, the distance between horizontal lines in the list zone reflects changes in the point size.

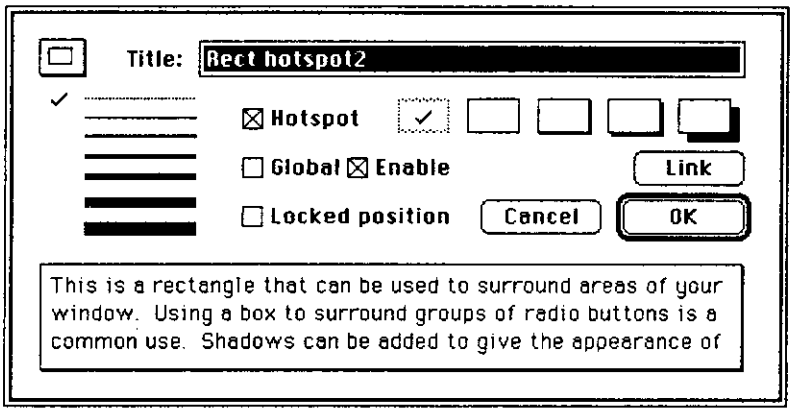
Rectangles

Rectangles are usually used to visually group together other zones, such as sets of radio buttons. Rectangles can have a shadow, can be drawn with different line thicknesses, and are supported in Windows, Dialogs, and Alerts.

New in Prototyper 3.0, you may define a rectangle zone to act as a HotSpot. A HotSpot may be invisible, and can be linked to evoke any number of additional actions when your prototype is running.

To set additional attributes for the rectangle zone:

1. **Double-click the rectangle zone in the Worksheet.**



2. **Enter the title.**

   The title appears in the source code to identify this particular zone.

3. **To give the rectangle a shadow: Select the icon designating the type of shadow.**

   This creates a shadow effect to the right and bottom of the rectangle.

4. **To change the line thickness: Click on the desired thickness.**

5. **To make the rectangle a HotSpot: Click the HotSpot check box.**

6. **To have the HotSpot 'remember' it's enable status: Click the Global Check Box.**

   A global HotSpot will have it's enable/disable state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

7. **To link a HotSpot to evoke any number of actions in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in its own chapter called "Linking". In summary, you can link hotspots to:

   - Open a window, dialog, alert, or demo window.
   - Close a window, dialog, alert, or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.

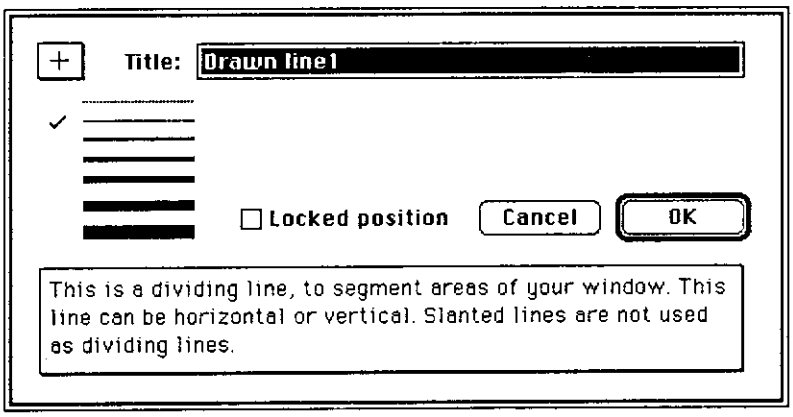
8. **Click OK or press Return.**

Lines

Lines are used as horizontal or vertical dividing lines in a window. Line direction can be changed from horizontal to vertical, or visa-versa, by dragging the handle toward the desired direction. The line snaps into place in the new direction. Lines are supported in Windows, Dialogs, and Alerts.

To set additional attributes for the line zone:

1. **Double-click the line zone in the Worksheet.**



2. **Enter the title.**

   This is the title appears in the source code to identify this particular zone.

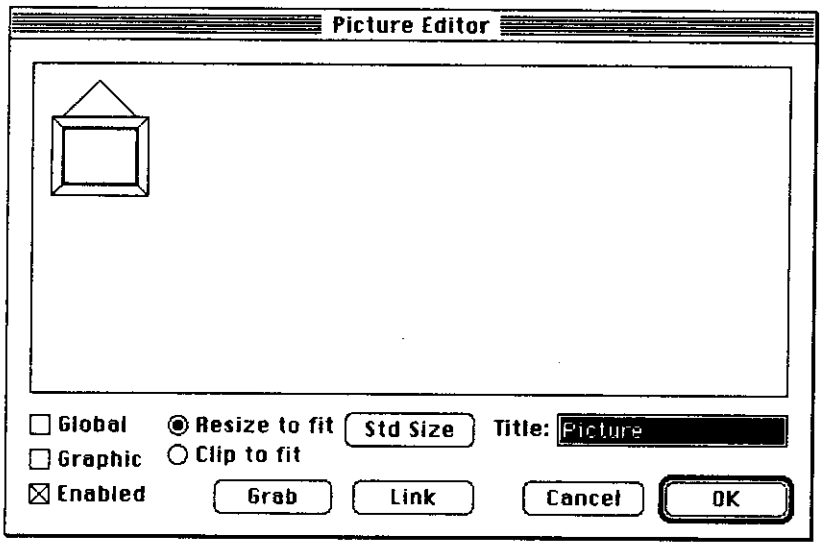
3. **To change the line thickness: Click the desired thickness.**
4. **Click OK or press Return.**

Pictures

Pictures can be imported into a window by using the Clipboard and Cut, Copy and Paste, or by loading resources from other applications or files. All other zone types can be placed on top of pictures, but Pictures cannot be placed on other zone types.

To place a picture in a window:

1. **Double-click the picture zone in the Worksheet.**



   This displays the Picture Editor, which is used to copy pictures from the Clipboard into a window.

2. **Enter the title of the picture.**

   This name appears in generated source code to identify the zone.

3. **Designate the size of the picture:**

   Before placing the picture, designate how the picture should be displayed in the window.

4. **Click Resize to Fit, Clip to Fit, or Default Size.**

   Resize to Fit causes the picture to be resized to fit into the area designated as the picture zone in the window.

   Clip to Fit causes the picture to copy as the exact size of the original picture in the Clipboard. But, if the zone area is smaller than the picture, only a portion of the picture is visible in the window.

   The Standard Size button sets the picture zone size to the exact original size of the picture being copied.

   Note: Pictures in dialogs and alerts can only be designated to Resize to Fit.

5. **Copy a Picture**

   To copy a picture from the Clipboard:

   Note: To copy a picture to a window, the picture must be in the Clipboard. The picture can be copied from another application or from the scrapbook. Copy the picture using the standard Macintosh commands. (If you are not familiar with the Clipboard, refer to the Macintosh manual before continuing.)

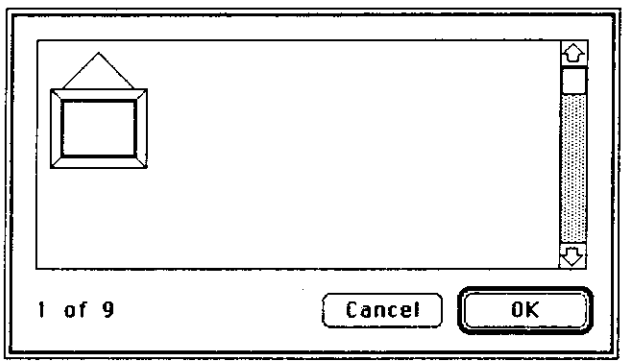
   1. Copy a picture to your Clipboard.
   2. Choose Paste from the Edit menu.

   The picture in the Clipboard is placed in the Picture Editor. It may be removed from the Picture Editor, and copied back to the Clipboard, by selecting Cut from the Edit menu.

6. **Grab a Picture**

   To grab a picture:

   1. Click the Grab button.



This displays a picture grabber similar to the icon grabber. The picture grabber stores the pictures that have been loaded into the prototype using the Load Resources command or that have already been created.

2. Scroll through the pictures.

3. Click OK and the picture moves to the Picture Editor.

7. **To have the picture 'remember' it's enable status: Click the Global Check Box.**

A global picture will have it's enable/disable state saved when the window is closed, and restored if the window is reopened during the execution of the application or prototype.

8. **To link a picture to evoke an action in your prototype: Click the Link button.**

This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in its own chapter called "Linking". In summary, you can link pictures to:

- Open a window, dialog, alert, or demo window.
- Close a window, dialog, alert, or demo window.
- Disable and Enable menus.
- Disable and Enable individual menu commands.
- Disable and Enable Zones and Custom Controls.
- Open Page Set-up and Printing Dialogs.
- Open File and Save Dialogs.
- Check or Toggle menu commands.
- Select Radio Buttons.
- Play Sounds.

Note: Pictures in windows & dialogs can be linked to open windows, dialogs and alerts. Pictures in alerts cannot be linked; they may only close the alert.

9. **Click OK or press Return.**

Plugin Buttons

Plugin buttons are supported in Windows, Dialogs, and Alerts, if installed with Prototyper's Plugin Installer. They can be used to customize the behavior (and "look") of a standard button, extending its functionality in the interface in any way desired by the writer of the Plugin.

Like standard buttons, Plugin Buttons act instantaneously when clicked, and reset to their normal state as soon as the mouse button is released.

To set additional attributes for a Plugin Button zone:

1. **Double-click the Plugin Button zone in the Worksheet.**
2. **Enter the title.**

   The title appears in generated source code to identify this particular zone.
3. **To disable a Plugin Button: Click the Enabled, not dimmed Check Box.**
4. **To link a Plugin Button to evoke an action in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in the later chapter "Linking". In summary, you can link Plugin Buttons to:

   - Open a window, dialog, alert, or demo window.
   - Close a window, dialog, alert, or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.
5. **To close the dialog: Click OK**

Plugin Toggles

Plugin toggles are supported in Windows, and Dialogs, if installed with Prototyper's Plugin Installer. They can provide a customized two-state button, extending its functionality in the interface in any way desired by the writer of the Plugin.

Like standard check boxes, Plugin Buttons toggle between states when clicked, and remain "latched" into the new state when the mouse button is released.

To set additional attributes for a Plugin Toggle zone:

1. **Double-click the Plugin Toggle zone in the Worksheet.**
2. **Enter the title.**

   The title appears in generated source code to identify this particular zone.
3. **To disable a Plugin Toggle: Click the Enabled, not dimmed Check Box.**
4. **To designate as initially selected: Click the Initially selected Check Box.**

   Setting a Plugin Toggle as initially selected causes it to start out in the selected state when the prototype is run.
5. **To link a Plugin Toggle to evoke an action in your prototype: Click the Link button.**

   This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in the later chapter "Linking". In summary, you can link Plugin Buttons to:

   - Open a window, dialog, alert, or demo window.
   - Close a window, dialog, alert, or demo window.
   - Disable and Enable menus.
   - Disable and Enable individual menu commands.
   - Disable and Enable Zones and Custom Controls.
   - Open Page Set-up and Printing Dialogs.
   - Open File and Save Dialogs.
   - Check or Toggle menu commands.
   - Select Radio Buttons.
   - Play Sounds.
6. **To close the dialog: Click OK**

Plugin Gauges

Plugin gauges are supported in Windows if installed with Prototyper's Plugin Installer. They can provide a customized multi-state control, extending its functionality in the interface in any way desired by the writer of the Plugin.

Like scroll bars, Plugin Gauges allow continuous variation over a range between pre-defined maximum and minimum values.

Research shows that many people find it easier to use such analog indicators to measure relative quantities than an equivalent digital display.

To set additional attributes for a Plugin Gauge zone:

1. **Double-click the Plugin Gauge zone in the Worksheet.**
2. **Enter the title.**

   The title appears in generated source code to identify this particular zone.
3. **To disable a Plugin Gauge: Click the Enabled Check Box.**
4. **Set the Maximum and Minimum values for the Plugin Gauge.**

   The highest possible value of a Plugin Gauge, in arbitrary units, is set using the Maximum value editable text box. The lowest possible value of a Plugin Gauge, again in arbitrary units, is set using the Minimum value editable text box.
5. **Set the initial value for the Plugin Gauge.**

   The initial value of a Plugin Gauge can be set using the Start At value. This is a relative number of units to offset the gauge from its lower limit.
6. **Set Large and Small Stepping rates.**

   Stepping rates are adjusted with the Small Step and Large Step increments used by the Plugin Gauge. The values used are relative to the Maximum and Minimum values set previously. The two rates are direct analogs of those used with scroll bars.

   The design of the Plugin Gauge determines how an individual gauge is controlled by the user when the prototype is used.
7. **To set the Plugin Gauge to its default width: Click the Default Width button.**

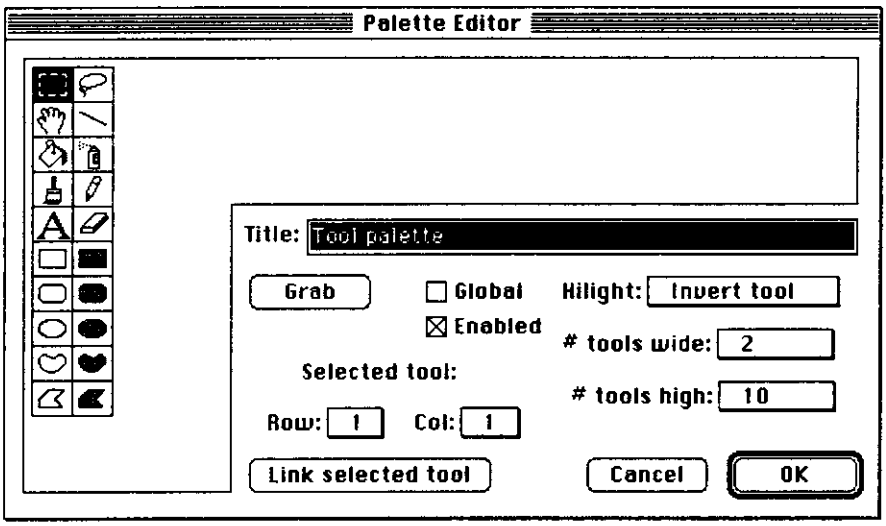
   If you have created a Plugin Gauge wider than 16 pixels, you can quickly resize it to the standard width without changing the size of the zone in the Worksheet manually.
8. **Click OK or press Return.**

Palettes

Palettes allow you to simulate the tool palettes often found in the Worksheets of graphics programs and other applications with a variety of data-manipulation tools. Any number of rows and columns can be created in a palette, and several palette designs are built into Prototyper already. If you wish to use your own palette design, you will first need to create a picture of the palette in a paint program. Make sure when creating a new palette design that all of the tools have the same height and width. Each tool can be linked to evoke a number of actions in your prototype when selected by the user.

To edit a palette zone:

1. **Double-click the Palette zone in the Worksheet.**



This displays the Palette Editor, which is used to grab Palette pictures from the Clipboard , and to adjust the behavior of the palette.

2. **Enter the title of the Palette.**

This name appears in generated source code to identify the zone.

3. **Paste a Picture**

To paste a palette picture from the Clipboard:

Note: To paste in a picture to a palette zone, the picture must already be in the Clipboard. The picture can be copied from another application or from the scrapbook. Copy the picture using the standard Macintosh commands. (If you are not familiar with the Clipboard, refer to the Macintosh manual before continuing.)

1. Choose Paste from the Edit menu.

The picture in the Clipboard is placed in the Palette Editor. It may be removed from the Palette Editor, and copied back to the Clipboard, by selecting Cut from the Edit menu.

4. Grab a Picture

To grab a picture:

1. Click the Grab button.

This displays a picture grabber similar to the icon grabber. The picture grabber stores pictures that have been loaded into the prototype with the Load Resources command or that have already been created.

2. Scroll through the pictures.

3. Click OK and the picture moves to the Picture Editor.

5. Adjust the # tools wide and # tools high settings correctly for your palette design.

Set the number of rows of tools and the number of columns of tools in the palette. This allows Prototyper to determine which area to highlight when a tool is selected.

6. To link an individual palette tool to evoke an action in your prototype:

1. Select a tool within the palette using the Row: and Col: Pop-up menus underneath the Selected Tool label.

To select the first tool in the palette, choose Row 1 and Col 1. To choose the tool from the second row, third column of your palette, choose Row 2 and Col 3. The selected tool will be highlighted in the window.

2. Click the Link selected tool button.

This displays the Linking dialog, which allows you to perform a number of linking operations. The Linking dialog and its capabilities are described in its own chapter called "Linking". In summary, you can link palette tools to:

- Open a window, dialog, alert, or demo window.
- Close a window, dialog, alert, or demo window.
- Disable and Enable menus.
- Disable and Enable individual menu commands.
- Disable and Enable Zones and Custom Controls.
- Open Page Set-up and Printing Dialogs.
- Open File and Save Dialogs.
- Check or Toggle menu commands.
- Select Radio Buttons.
- Play Sounds.

3. Select each of the tools in succession, and link each one as you wish.

7. **To change the highlighting of the selected tool: Choose a highlighting method from the Hilight Pop-up menu.**

   The default highlighting method is to simply invert the tool. You may experiment with other methods of highlighting by choosing them from the list.

8. **Click OK when you are finished to return to the Worksheet.**

HotSpots

HotSpots allow an invisible rectangular area in your Worksheet to link to any number of actions in your prototype when clicked by the user. Please refer to the description of Rectangle zones, above, for more information.

Creating Menus

4. Creating Menus

Menu Editor 4-2
- Editing Menus 4-3
- Standard Macintosh Menus 4-4
- Adding New Menu Titles to the Menu Bar 4-5
- Editing Menu Titles 4-6
- Deleting Menus from the Menu Bar 4-7
- Rearranging Menus 4-8
- Cutting, Copying and Pasting Menus 4-9

Commands 4-10
- Adding New Commands 4-11
- Disabling Commands 4-12
- Adding Separator Lines Between Commands 4-13
- Changing Font Styles for Commands 4-14
- Designating Commands to be Initially Checked 4-15
- Designating Command Keys 4-16
- Designating Icons for Commands 4-18
- Editing Commands 4-19
- Deleting Commands from Menus 4-20
- Rearranging Commands 4-21
- Cutting, Copying and Pasting Commands 4-22

Hierarchical Menus 4-23

Linking Menu Commands 4-25

Creating Menus

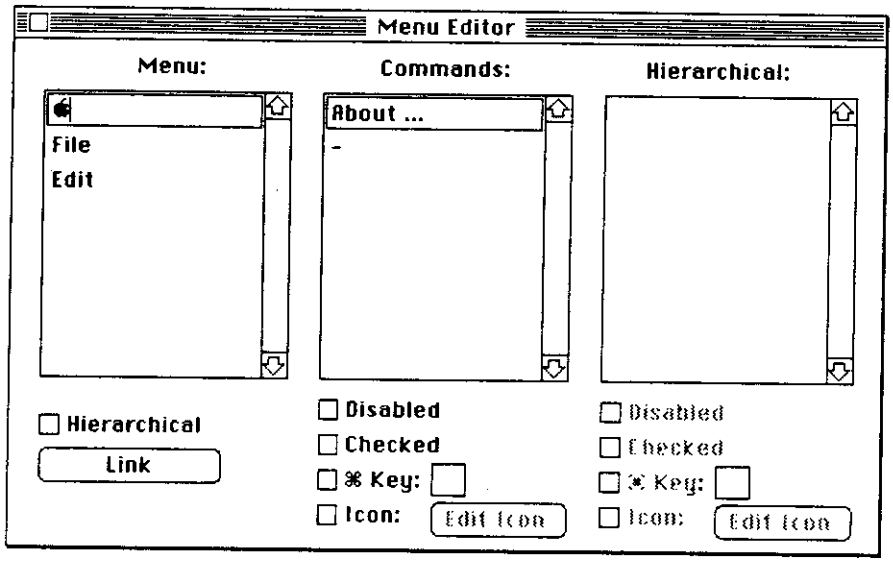
Menu bars list options that are available for the user to choose from at any given time. The user can browse through the menu bar to choose from the available options. With Prototyper it's easy to build a menu bar and link commands to perform various actions. Prototyper already includes the standard Apple, File and Edit menus and their commands in every prototype.

The menu structure consists of the menu bar, which is always visible at the top of the screen and displays the menu titles, the menus, and the commands for each menu. Menus consist of the menu title and the commands under the title. When we refer to menus in this chapter, we are referring collectively to the entire menu, including the title, all the commands under it, and any hierarchical sub-menus.

In this chapter, you will learn how to use the Menu Editor to build menus for your prototype.

Menu Editor

The menu button in the Prototype Window leads to the Menu Editor. The Menu Editor contains a Menu column, a Commands column, a Hierarchial Commands column, and various check boxes for specifying additional attributes for the commands. The Menu column contains the menu titles for the prototype. The Commands column contains the commands for each menu title. When a menu title is selected in the Menu column, the commands for the selected menu appear in the Commands column. If the selected Command is Hierarchial, it's Hierarchial Sub-Commands will be displayed in the Hierarchial column.



Editing Menus

At any point in the development of a prototype, the menu bar, menus and commands may be edited or changed.

To open the Menu Editor:

1. **Click the Menu button in the prototype window.**

   This displays the Menu Editor for the prototype. Change, modify, or edit the menus and commands.

2. **Click the close box to close the Menu Editor.**

   The Menu Editor can remain open while other windows are active. Click the Menu Editor window or the Menu button in the Prototype Window to make it active.

Standard Macintosh Menus

If you are creating a Macintosh application, you should include the standard Apple, File and Edit menus in the menu bar. Prototyper automatically provides these menus along with their standard commands.

When a menu is selected, a rectangle outlines the menu title in the Menu column. Selected menus are also available for viewing with Prototyper's instant simulation mode. The selected menu appears to the right of the Prototyper menu bar. By pressing on the simulated menu title on the Prototyper menu bar, the menu and its commands are displayed. This provides instant feedback about how each menu and its commands will look when the prototype is used.

Every time the Menu Editor is opened, the Apple menu in the Menu column is initially selected, and appears to the right of the Prototyper menu bar. The Apple menu title in the Menu column is surrounded by a rectangle. The Commands column displays all the commands under the Apple menu.

To display the commands for the File and Edit menus:

1. **Select the File menu title in the Menu column.**

   Notice that the File menu appears to the right of the Prototyper menu bar, replacing the Apple menu. The commands for the File menu are displayed in the Commands column and under the simulated File menu in the menu bar. The Edit menu and its commands can be seen by selecting the Edit menu title in the Menu column.

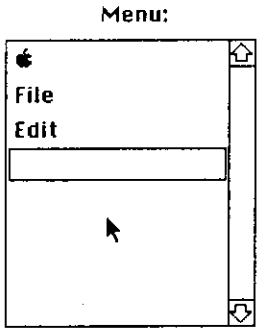
   Note: For more information on standard System Menus, see Inside Macintosh, Vol. I.

Adding New Menu Titles to the Menu Bar

In addition to the standard Macintosh system menus, new menus can be added to the menu bar. Menu titles are one-word descriptions of the commands under them and the first letter should always be capitalized. For example, a Font menu might contain different styles and point sizes.

To create new menus:

1. **Press the Tab key to select the open space under the last menu title in the Menu column.**



2. **Enter the new menu title.**

   If you want to create a blank space in the menu bar, press the space bar.

3. **Press Return or Tab after each new menu title.**

   This advances the cursor to the next open space.

Editing Menu Titles

The text of menu titles can be edited using standard Macintosh editing methods.

To edit a menu title:

1. **Select the menu title to be edited.**
2. **Click to insert the cursor, click and drag to select text, or double-click to select a single word.**
3. **Enter the new text.**

Deleting Menus from the Menu Bar

Menus in the Menu column respond to Cut and Clear from the Prototyper Edit menu. Cutting or clearing a menu title also cuts or clears all the commands for the menu.

Note: Commands created for a menu title become part of the entire menu and always remain with the menu title they were originally assigned to.

To delete a menu title:

1. **Select the menu title from the Menu column that represents the menu to be deleted.**

   A rectangle appears around the menu title indicating that it is the selected menu.

2. **Choose Cut or Clear from the Prototyper Edit menu.**

   Cut places the entire menu (including the menu title and commands) on the Clipboard, while Clear completely eliminates the entire menu from your prototype. Clear is NOT undo-able. Cut can be reversed by immediately doing a Paste.

   Note: When the prototype Edit menu is selected in the Menu Column of the Menu Editor. Prototyper's own Edit menu command key equivalents are effectively disabled. This is because these equivalents are duplicated in the current menu bar - once in Prototyper's Edit menu and once in the simulated menu. Macintosh's Menu Manager works from right to left when processing command key equivalents.

Rearranging Menus

In addition to editing or deleting a menu from the menu bar, the order of menus in the menu bar can be changed. Changing the menu order can be accomplished using one of two methods. Use the grabber hand to drag the menu to a new location in the menu bar, or cut, copy and paste. When menus are moved or repositioned using either of these methods, the commands for each menu move to the new location also.

Note: Commands created for a menu title become part of the entire menu and always remain with the menu they were originally assigned to.

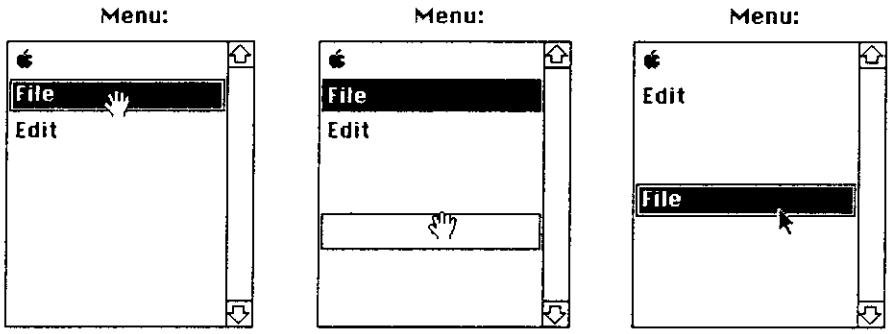
To rearrange menus:

1. **On an unselected menu title, press with the mouse on the desired menu title in the Menu column and keep the mouse button depressed.**

   The pointer changes to a grabber hand and the menu title is highlighted. Moving the hand up or down creates a positioning rectangle. The positioning rectangle indicates where the menu title is being repositioned.

   Note: To move a menu title that is already selected, select another title, then reselect and drag the original title.

2. **To place the menu between two other menus, move the positioning rectangle between the two menus.**



When the desired location in the menu bar has been reached:

3. **Release the mouse button and the menu title snaps into place, moving the menus beneath it down to the new location.**

Cutting, Copying and Pasting Menus

Menus can be Cut or Copied and Pasted into a new location. They can be pasted into an open space or on top of another menu. If a menu is pasted on top of another menu, it replaces the original menu completely. Cutting or Copying a menu title also copies or cuts all the commands for the menu.

To change the order of menus using Cut, Copy and Paste:

1. **Select the desired menu title in the Menu column.**
2. **Choose Cut or Copy from the Edit menu.**

   Cut removes the entire menu from the prototype, while Copy places a copy of it, complete with its commands, in the Clipboard.
3. **Select the desired location for placing the menu.**

   A rectangle is displayed around the menu or open space to show which space is selected for pasting the menu into.
4. **Choose Paste from the Edit menu.**

Commands

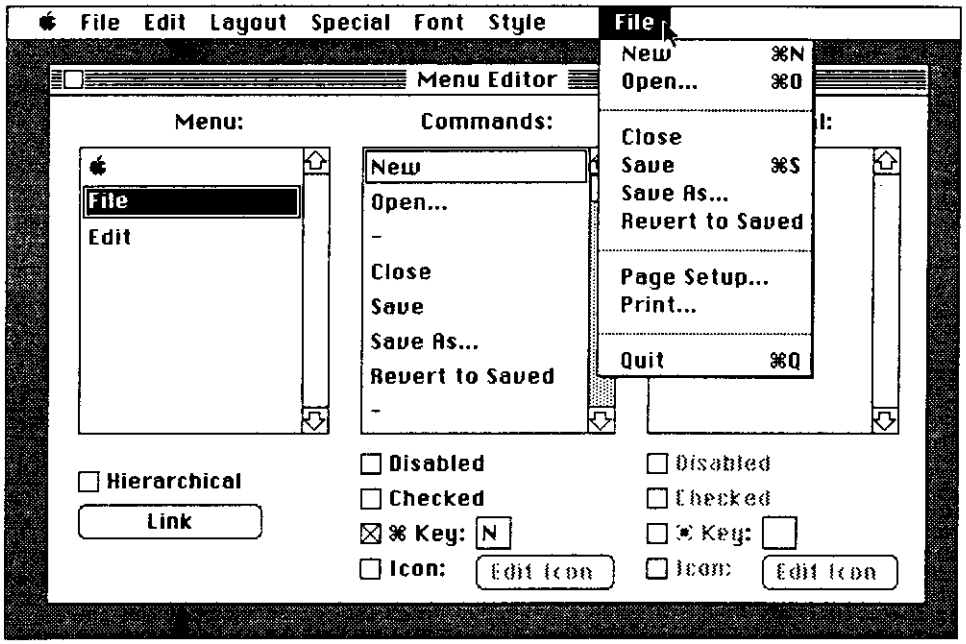
Standard Macintosh Commands

As mentioned at the beginning of the chapter, Prototyper provides the standard commands for the Apple, File and Edit menus. There is no need to create these in the Commands column.

To examine the commands for the standard Macintosh menus:

1. **Select the desired menu title from the Menu column.**

   A list of commands under the selected menu appears in the Commands column.



Note: At any time during the process of building a menu bar and menus, the commands can be observed by selecting the simulated menu at the right of the Prototyper menu bar.

Adding New Commands

Use the Menu Editor to create new commands for new menus or to add new commands to the existing Macintosh menus.

Commands should be concise, preferably one word, with the first letter capitalized. If more than one word is used, the first word only should be capitalized.

Apple Computer's Human Interface Guidelines recommend that commands should be either verbs or adjectives. Use verbs or verb phrases to show the user what can be done. For example, Cut, Run, or Set location. Use adjectives to specify attributes of a selected object, such as changing the font or style. A single menu can contain both verbs and adjectives, but verbs and adjectives should be grouped together and be separated by a dotted line, called a separator line.

Frequently used commands should be placed at the top of the menu to provide easy access.

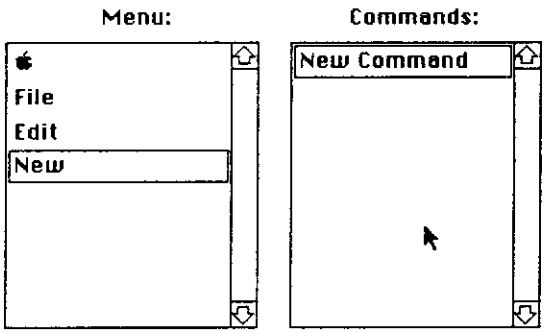
To create new commands for menus:

1. **Select the menu title from the Menu column that you want to create commands for.**

   This places the selected menu to the right of the Prototyper menu bar and creates a rectangle around the menu title to indicate that it is the selected menu.

2. **Select the first open space in the Commands column.**

3. **Enter the command.**



Note: In accordance with Apple Computer's Human Interface Guidelines, commands that open windows and dialogs should always be followed by an ellipsis (...) in the menu.

4. **Press Return or Tab after each command to advance the cursor to the next open space.**

   Notes: Notice that when the command is entered, it appears under the simulated menu in the Prototyper menu bar. When the list is full, it will become necessary to click in the down arrow of the scroll bar to create a new menu item.

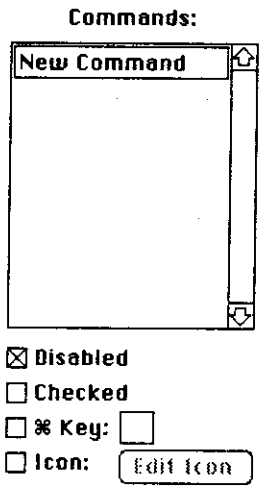
Disabling Commands

Some commands in a menu cannot always be chosen while using the application or prototype. These commands are termed disabled and appear grey in the menu. If the user tries to choose the command, it is not highlighted. With Prototyper you can signify disabled commands. When the prototype is run, the disabled command appears grey in the menu.

Disabled commands can be enabled and disabled as the prototype is run with new linking features. These features are described in the Linking chapter.

To disable a command:

1. **Select the command.**
2. **Click the check box for disabled.**



Adding Separator Lines Between Commands

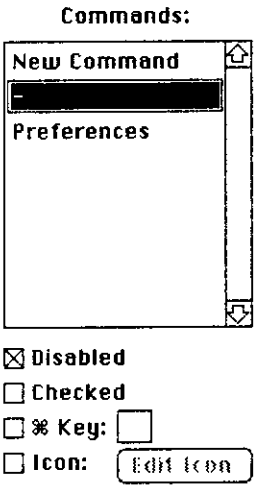
Separator lines are used to divide commands into logical groups. They are usually placed between groups of verb commands and adjective commands. A separator line is actually a disabled command.

To place a separator line in the Commands column to separate two commands:

1. **Select an open space.**

   If there isn't an open space between the two commands, drag an open space between the commands from below. See Reordering Commands.

2. **Press the hyphen (-) key.**
3. **Click the Check box for Disabled.**



4
Creating the Menu Bar

Changing Font Styles for Commands

On occasion menu commands are assigned text styles for emphasis. The most obvious and most common application of styles to commands is exemplified in Prototyper's own Style menu. In the Style menu, the Italic command is displayed in italic, Bold in bold, etc. Your prototype, in some rare instances, may require you to change the text style on a command for a certain type of emphasis.

To change the font style of a command:

1. **Select the command.**
2. **Choose the desired style from the Style menu.**

   Prototyper's Style Menu lets you choose from Plain Text or any combination of:

   - Bold
   - Italic
   - Underline
   - Outline
   - Shadow

   Styles can only be changed for commands. You cannot change the styles of menu titles.

Designating Commands to be Initially Checked

Certain commands work interdependently and are grouped together either as mutually exclusive groups or as accumulating groups.

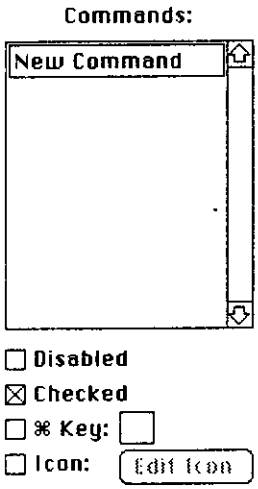
In a mutually exclusive group, only one command is in effect at one time. The command is preceded by a check mark. An example is Prototyper's group of alignment commands in the Layout menu, where at any one time only one command is checked.

In an accumulating group, any number of commands can be in effect at the same time. An example of this is the Prototyper Style menu, in which you can choose any combination of Bold, Italic, Underline, Outline, or Shadow text styles.

Prototyper allows commands to be initially checked when the prototype is run. Commands can be made to toggle or work interdependently with each other, by using the linking features available to each menu.

To designate a command to be initially checked when the prototype is run:

1. **Select the command.**
2. **Click the check box for Checked.**



4

Creating the Menu Ba

Designating Command Keys

Some commands, especially often-used commands, may be given keyboard equivalents. This means the command can be chosen from the keyboard as well as from the menu. The keyboard equivalent is called a command key and follows the cloverleaf symbol in the menu. A command key can be assigned for any character key on the Macintosh keyboard.

To maintain consistency across Macintosh applications, Apple Computer reserves certain command keys which should not be used for any other command. They are listed below.

Apple menu

- Command - ? (Help)

File menu

- Command - N (New)
- Command - O (Open)
- Command - S (Save)
- Command - Q (Quit)

Edit menu

- Command - Z (Undo)
- Command - X (Cut)
- Command - C (Copy)
- Command - V (Paste)

The command keys for the Style menu are less strictly reserved. If your prototype has a Style menu, these keys are reserved. If your prototype doesn't have a Style menu, these keys can be used for other purposes if necessary.

Style menu

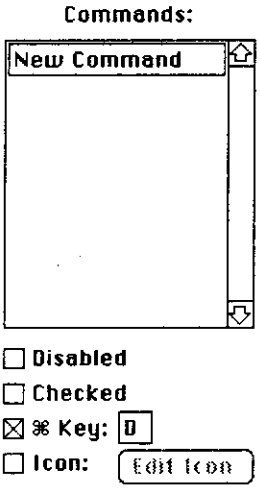
- Command - P (Plain)
- Command - B (Bold)
- Command - I (Italic)
- Command - U (Underline)

To designate a command key:

1. **Select the command.**
2. **Click the check box for the Command Key.**

   (This is the check box with the cloverleaf symbol followed by the word Key.)

   Note: Do not give more than one command the same command key. If you give the same command key to two commands, the last command in the menu bar, from left to right, is chosen when the command key is used.
3. **Click the check box next to the Command Key check box.**
4. **Enter the designated command.**



4

Creating the Menu Ba

Designating Icons for Commands

It is possible to place icons in the menu as a graphical representation of the command. This feature is rarely used in most applications, but may be useful in special circumstances.

To create an icon for a command:

1. **Select the command.**

   The icon appears on the left side of the column, preceding the command.

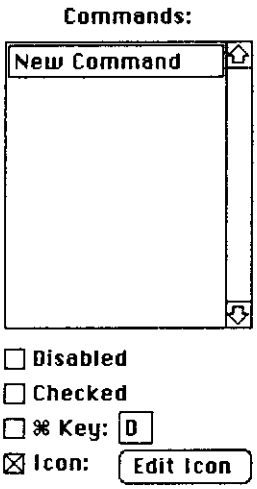
2. **Click the check box for Icon.**

3. **Click on the Edit Icon button.**

   This displays the Icon Editor. (See "Icon" in the "Creating Zones" chapter for detailed instructions on the Icon Editor.)

   Note: When using the Icon Editor to create an icon for a command, you will not be able to create a highlighted version of the icon. This is not necessary, since the icon inverts automatically when the command is chosen.

4. **Create or Load an icon using the Icon Editor. Click OK to return to the Menu Editor.**



Editing Commands

The text for commands can be edited using standard Macintosh editing methods.

To edit a command:

1. **Select the command to be edited.**

   This highlights the entire command. If you are replacing the entire command, enter the new title.

2. **Click to insert the cursor, click and drag to select text, or double-click to select a single word.**

3. **Enter the new text.**

Deleting Commands from Menus

Commands in the Commands column respond to Cut and Clear from Prototyper's Edit menu.

To delete a command:

1. **Select the desired command from the Commands column.**

   A rectangle appears around the command to indicate it is selected.

2. **Choose Cut or Clear from the Edit menu or press the delete or backspace key.**

   Cut places the command on the Clipboard, while Clear completely eliminates the command from your prototype. Clear is NOT undo-able. Cut can be immediately reversed by choosing Paste.

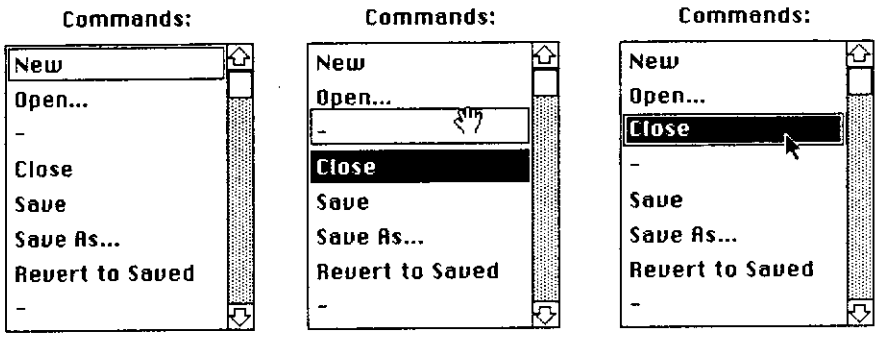
Rearranging Commands

Changing the order of commands is accomplished in the same manner as changing the order of menus.

To rearrange commands:

1. **Press on the desired command in the Commands column with the mouse and keep the mouse depressed.**

- The pointer changes to a grabber hand and the command is highlighted. Moving the hand up or down creates a positioning rectangle. The positioning rectangle indicates where the command is being repositioned.



2. **To place the command between two other commands, move the positioning rectangle between the two existing commands.**

   When the desired location in the Commands column has been reached:

3. **Release the mouse and the command snaps into place, moving the commands beneath it down one space each.**

4
Creating the Menu Ba

Cutting, Copying and Pasting Commands

Commands respond to Cut, Copy and Paste. Cut or Copy a command from one menu and Paste it under another menu. It can be pasted into an open space or on top of another command. If a command is pasted on top of another command, it replaces the original command completely.

To change the order of commands using Cut, Copy and Paste:

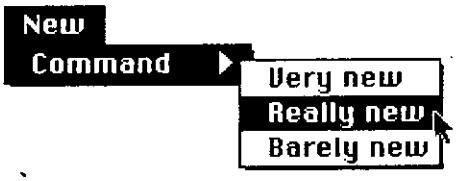
1. **Select the desired command in the Commands column.**
2. **Choose Cut or Copy from the Edit menu.**

   Cut removes the command from your prototype and places it in the Clipboard, while Copy places a copy of it in the Clipboard without removing it.
3. **Select the desired location for placing the command.**

   A rectangle is displayed around the command or open space to show what space is selected for pasting in the command.
4. **Choose Paste from the Edit menu.**

Hierarchical Menus

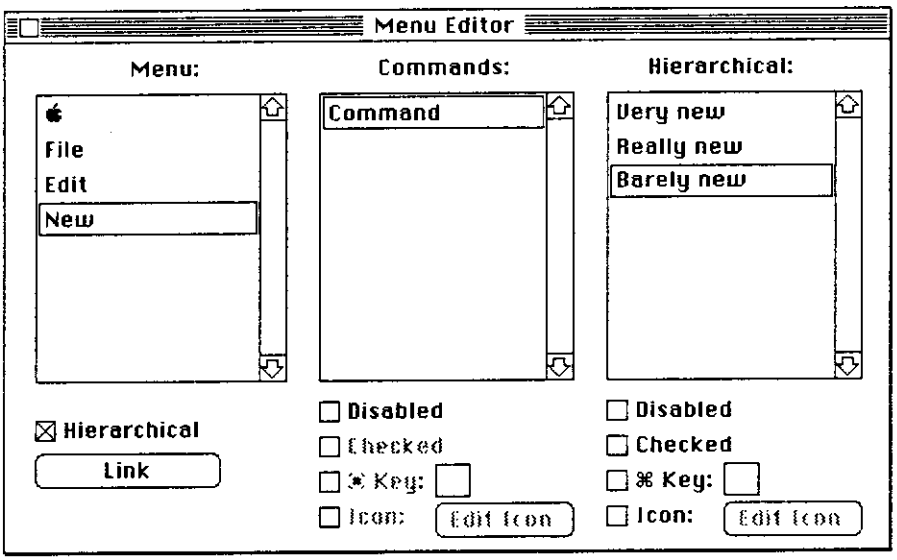
Hierarchical menus allow you to make a menu item become the title of another sub-menu. When a menu item forms the title of a sub-menu, it is followed by a small black arrow. These sub-menus do not appear until after a short delay, so they will not flash (appear and reappear) as users browse through menus.



Sub-menus should only be used for lists of related items, such as font styles, baud rates, or font sizes, etc. Apple recommends that only one level of hierarchical menus should be used, although the Macintosh can support more than one level. Prototyper currently only supports the recommended single level.

To make a standard menu item into a hierarchical menu:

1. **Select the desired menu item in the Command column of the menu editor that you wish to make into a sub-menu title.**



Rename the command if necessary or create a new item by dragging the rectangle to the desired position and entering the new title.

2. **Click the check box labeled Hierarchical.**

   Notice how the series of check boxes under the commands column are now disabled and the series of check boxes under the Hierarchical column are now enabled.

   You can now edit the hierarchical menu items in exactly the same way you edit standard menu commands as described earlier in this chapter.

   Note: Since the menu item in this instance is a sub-menu title for a series of hierarchical menu commands, the following attributes cannot be applied to the sub-menu title: Checked, Command key equivalent, and Icon.

3. **To name the hierarchical menu items: Enter the items in the Hierarchical column in the same manner as in the Commands column.**

   Look at the simulated menu to the right of the Prototyper menu bar. Notice how the hierarchical menu title now has a small black triangle pointing to the right to indicate that more is available after a brief delay.

   Note: If you are using the standard 9-inch Macintosh screen, a menu simulated while in the menu editor will produce hierarchical menus that appear to the left hand side of the main menu, rather than disappearing off-screen to the right. This is a feature of the Macintosh. If you wish to see the hierarchical menus appear to the right, select Run from the Special menu and try again.

4. **Click the window's close box to exit the menu editor or, click the prototype window and underlying Worksheets and proceed with your Prototyper session.**

Linking Menu Commands

Both hierarchical and menu commands can be linked to trigger a variety of actions in your prototype. However, the most common link you will use will most likely be the opening and closing of windows, dialogs, alerts, and demo windows.

Menu commands often lead to modal dialogs, such as Page Setup, to allow the user to set up parameters. These commands should be followed by an ellipsis (...) in the menu, to let the user know more information is required.

Linking capabilities are described in the Linking chapter following this chapter.

In summary, Prototyper allows you to link menu commands to:

- Open a window, dialog, alert, or demo window.
- Close a window or demo window.
- Disable and Enable menus.
- Disable and Enable individual menu commands.
- Open Page Set-up and Printing Dialogs.
- Open File and Save Dialogs.
- Check or Toggle menu commands.
- Play Sounds.

To link a menu command:

1. **Select the desired command in the Commands or Hierarchical column.**
2. **Click the Link button.**

   A Linking dialog box appears. The Linking dialog is described in the next chapter.
3. **Click the linking actions that the command will execute when chosen as the prototype is run.**

   A checkmark appears next to the link item to indicate that it is linked to a command. If you link more than one link item to a command, when the prototype is run, all windows open first, then dialogs, then alerts. For example, if you link a window and a dialog to a command, the window always appears before the dialog when the prototype is run.
4. **Click OK or press Return.**

Linking

5. Linking

The Linking Dialog 5-2
Linking Dialog Icon Buttons 5-3
Linking From a Menu Command 5-7
Linking From Zones 5-8
Creating a Link Map 5-9

Linking

Linking allows actions taken by the user of the prototype to trigger a variety of responses.

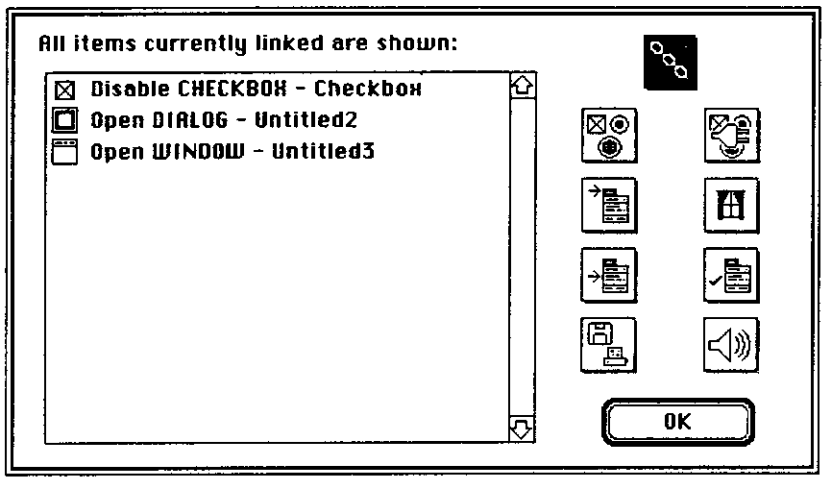
The most common linking activity for most prototypes is the opening and closing of windows, dialogs, alerts, and demo windows from menu commands or buttons in windows. However, Prototyper's extensive linking features support almost any imaginable combination of responses to a user action — for example, you may want pressing a button in a window to disable a menu, a menu command, or other zones in the same window. This type of linking capability is available through the Linking Dialog.

In this chapter you will learn how to link menu commands, buttons, check boxes, radio buttons, pop-up menus, icons, pictures, zones, and Plugin zones to perform various actions. In addition, you will learn how to create a link map text file.

The Linking Dialog

The Linking dialog is accessed through a Link button, which is available in the menu editor, and a variety of "Zone info" dialog boxes for selected zones. To perform any linking operations, the Linking Dialog dialog must be accessed.

When the Linking dialog appears, it displays in the list area all currently defined links for the selected zone or menu command. The icon selected while this list is displayed is called the Link icon.



You can perform linking operations to:

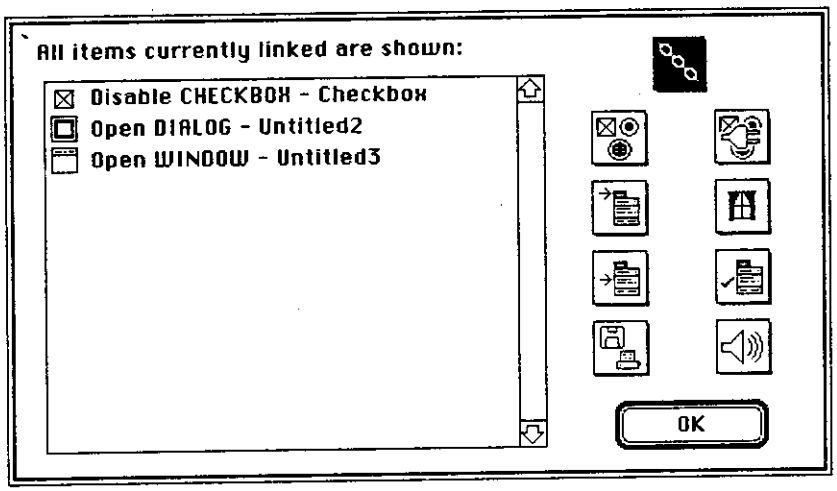
- Open a window, dialog, alert, or demo window.
- Close a window, modeless dialog, or demo window.
- Disable and Enable menus.
- Disable and Enable individual menu commands.
- Check or Toggle Menu Items.
- Disable and Enable zones.
- Open Page Set-up and Printing dialog boxes.
- Open File and Save dialog boxes.
- Play a sound.
- Enable/Disable, or Toggle a custom Plugin zone.
- Set a cursor delay time.

Linking Dialog Icon Buttons

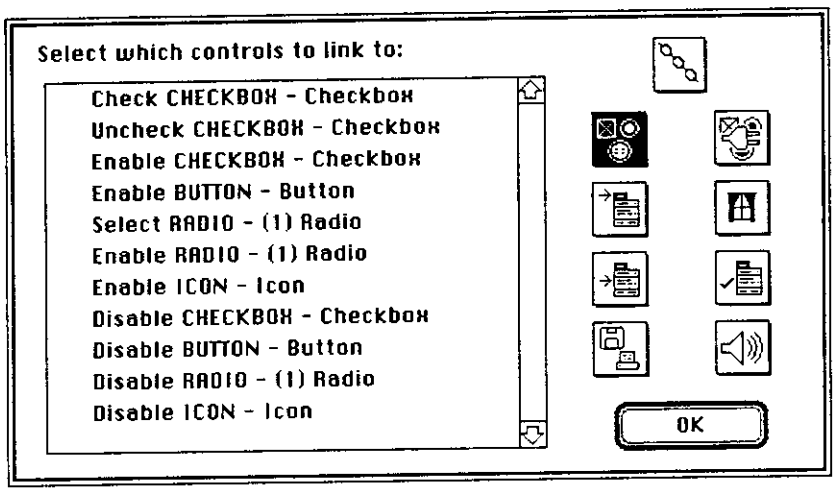
There are eight icon buttons in the Linking dialog that represent the various linking catagories that can be performed on the selection of the current zone or menu command. To activate a link, simply select it with the mouse.

You can choose from the following:

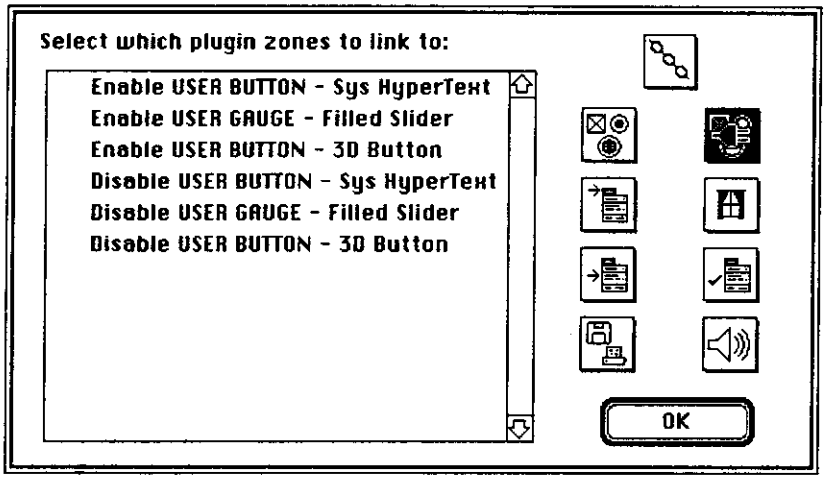
The chain icon will list all of the links currently active for this one or menu.



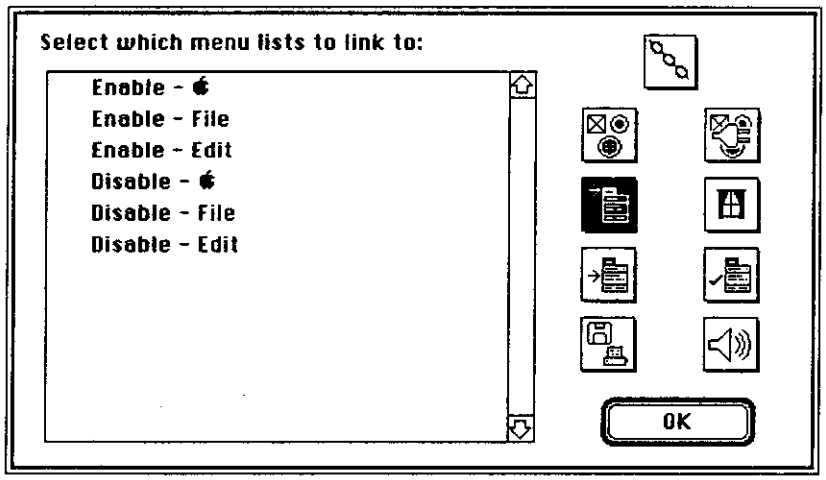
This list displays links to enable or disable, select or toggle standard zones when the link is executed.



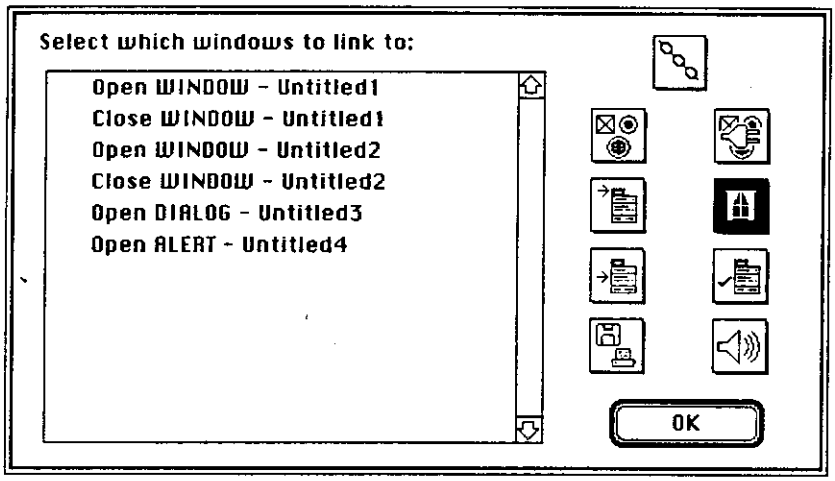
This list displays links to enable or disable, select or toggle custom controls when the link is executed.



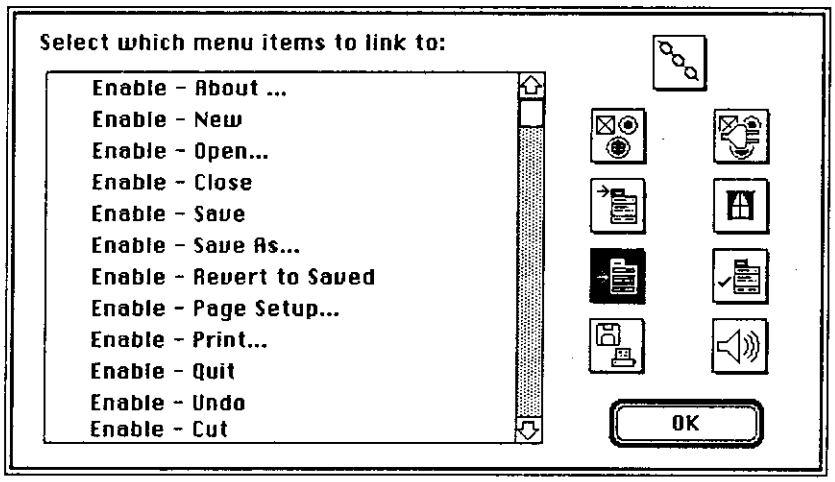
This list you can cause an entire menu and all of its menu items to be enabled or disabled when the link is executed.



Pressing this button makes Prototyper 3.0 display all of the available windows, dialogs, alerts, and demo windows. Using this list, you can cause these windows to be opened or closed when the link is executed.



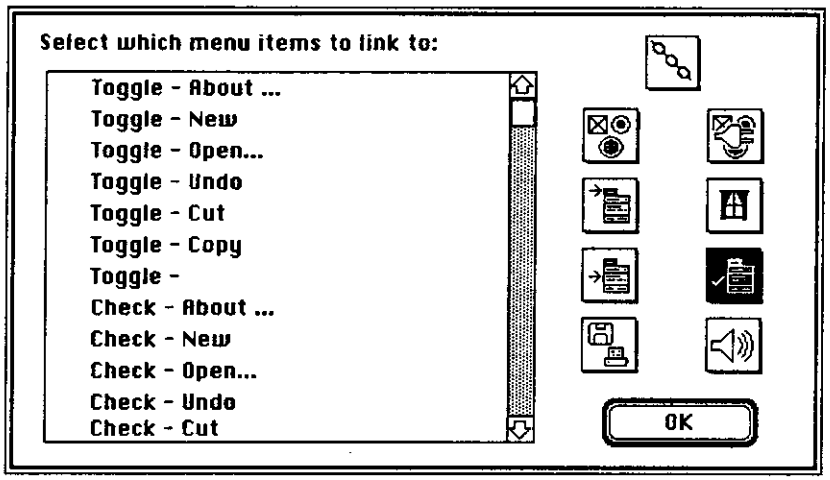
Pressing this button makes Prototyper 3.0 display all of the individual menu items (commands) in the menu bar. Using this list, you can cause any individual menu item to be enabled or disabled when the link is executed.



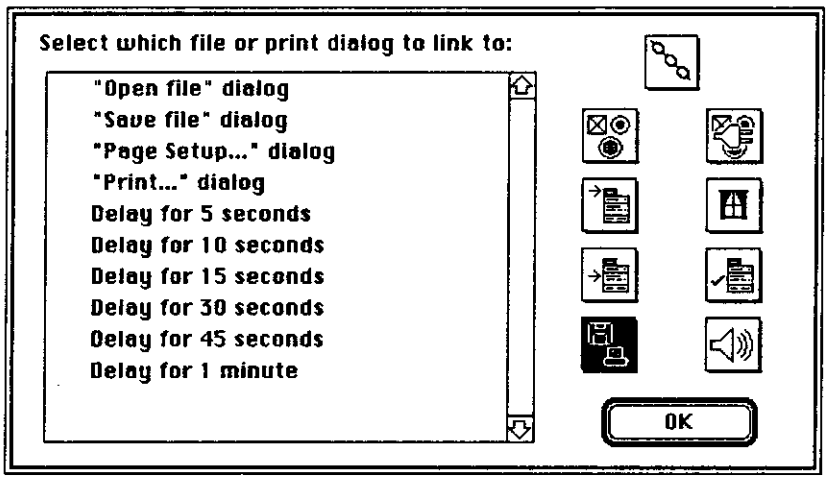
5

Linking

Pressing this button makes Prototyper 3.0 display all of the individual menu items (commands) in the menu bar. Using this list, you can cause any individual menu item to be checked or toggled when the link is executed.



This section allows selection of links to the standard Macintosh "Page Setup...", "Print...", "Open File...", , "Save File..." dialogs, or watch cursor. Selecting one of these causes the appropriate dialog to appear on the screen or the cursor to change for the selected time, when the link is executed.



Pressing the speaker button makes Prototyper 3.0 display all available sounds. Using this list, you can cause a sound to be played when the link is executed by the user.

Linking From a Menu Command

Menu items or commands are most commonly linked to open and or close windows, however, menu commands can be linked to perform all linking operations except those that operate on worksheet zones or plugins.

Note: There is no difference between the ability to link a standard menu command or a hierarchical menu command.

To link a menu command to evoke an action in your prototype:

1. **Click the Menu Button in the prototype window.**

   This displays the Menu Editor.

2. **Select the desired menu item, or hierarchical menu item.**

   The rectangle surrounds the selected item.

3. **Click the Link button.**

   This displays the Linking Dialog and all current links. Notice that the Link icon button is selected.

4. **Click the icon button for the type of link you wish to perform.**

   A list of all the available corresponding linkable items is displayed. If there is a check mark next to an item in the list, then it has already been linked.

5. **Click on the item you wish to link.**

   A check mark is displayed showing that it has been selected. You can check or uncheck as many items in the list as you wish.

6. **Click another icon button to perform more links or click the Link icon button to review all current links.**

7. **Click OK or press Return.**

   This returns you to the Menu Editor.

Linking From Zones

Zones are frequently used to link actions to application responses in a prototype. Linking is supported for most zone types, even though in some cases it may not always be appropriate to respond to a user's interaction with a zone by triggering an immediate action.

For instance, both check boxes and radio buttons support linking, but Apple's Human Interface Guidelines specifically reserve these controls for setting conditions for future actions, prohibiting instantaneous actions from resulting from interaction with them.

To link a zone to evoke an action in your prototype:

1. **Double-click the zone in the Worksheet, or select Zone info from the Info menu.**

   This displays the Zone Info dialog box.

2. **Click the Link button.**

   This displays the Linking Dialog and all current links. Notice that the Link icon button is selected.

   Note: In the case of Check Box and Plugin Toggle zones, the Link button in the Zone Info dialog is replaced by Link unselected and Link selected buttons. These buttons allow you to specify linking actions for both zone states independently.

3. **Click the icon button for the type of link you wish to perform.**

   A list of all the available corresponding items, zones, or windows is displayed. If there is a check mark next to an item in the list, it has already been linked.

4. **Click on the item you wish to link.**

   A check mark is displayed showing that it has been selected. You can check or uncheck as many items in the list as you wish.

5. **Click another icon button to perform more links, or click the Link icon button to review all links that have been made to respond to the menu command.**

6. **Click OK or press Return.**

   This brings you back to the zone info dialog.

7. **Click OK or press Return.**

   Note: If you have difficulty with zone linking when you try out your prototype, check that the zone is enabled. In Picture, Icon, and Small Icon zones, the Graphic check box effectively disables the zone, preventing links from being activated.

Creating a Link Map

The "Save Map..." option from the File menu lets you save a "Link Map" of the components of your prototype and its links, including menus, windows, zones, etc. The Map is a standard text document and can be opened and printed with any ordinary Macintosh word processor.

The Map file is especially useful for larger prototypes, where it is difficult to keep track of all the links. The Map document lists all the components of your prototype, allowing you to quickly see which component is linked to which. The Map also shows dialog position, height and width, and the position of all controls in the windows.

To save a map of your prototype's links:

1. **Select "Save Map..." from the File menu.**
2. **Enter a name for the Link Map document.**
3. **Click the Save or Cancel button.**

   If you select the Save button, the Link Map is saved as a text document.

Hard Disk:Sample Prototype Map

```
==============================================================================
*
* This is a map of all items in the prototype.
* File: Sample Prototype Map
* History: 7/19/90 Original by Prototyper.
* This file contains the map of all items and linkages.
*
==============================================================================


Menu list:   0 (Apple menu)
        Item:   About ...
                Linked to open a Modal Dialog named "About Dialog Box"
        Item:   -
                Disabled

Menu list:   File
        Item:   New
        Item:   Open...
        Item:   -
                Disabled
        Item:   Close
        Item:   Save
        Item:   Save As...
        Item:   Revert to Saved
        Item:   -
                Disabled
        Item:   Print Setup...
        Item:   Print
        Item:   -
                Disabled
        Item:   Quit
                Cmd Key Q

Menu list:   Edit
        Item:   Undo
```

5

Linking

Loading Resources & Printing

6. Loading Resources & Printing

What are Resources? 6-2
Loading Resources 6-3
Printing 6-6
    Printing Menus and Windows 6-6
    Printing to a document 6-6

Loading Resources

This chapter provides a brief overview of Macintosh resources and describes how Prototyper can load resources into your prototype to save you time.

You can load all standard Macintosh user interface resources from either prototypes or applications into your prototype. You can pick selectively from the common resource types, including menus, windows, dialogs, alerts, icons, pictures, and sounds; or you can simply load all available resources into your prototype at once.

Note: You can also install code resources that redefine or extend Macintosh interface definition code, like CDEF and WDEF resources, using the Prototyper Plugin Installer described later. See the section on the Plugin Installer.

What are Resources?

Resources are information structures that form the foundation of a Macintosh program. Resources are stored in a resource file, as an integral part of an application. Resources are distinguished by their type, (a four character code,) and by a two byte ID number. Resources may also have an optional name. When necessary, these resources are copied into memory by the Resource Manager, described further in Inside Macintosh.

Numerous disk-based resource types complement the memory-based structures of the Macintosh Toolbox. The Window resource, of type WIND, complements the Toolbox runtime Window record. A WIND resource provides a quick way to initialize a window's data structures, establishing its dimensions and appearance with little intervention.

Predefined Macintosh resources include menus, windows, dialogs, alerts, fonts, icons, etc. You may define your own resource types, and use them in the same way as predefined types (though there are restrictions on the range of ID numbers you may use — see Inside Macintosh for more information). The resource's modularity means that you can re-use resources from application to application.

Resources can be edited by any application which understands their structure. A resource editor like Apple Computer's ResEdit can alter the dimension of dialogs, change the text of buttons, relabel menu items, and so on. This is an important way to reduce the ost of localizing versions of applications for overseas markets. A distributor can change your application's error messages to the local language, without needing access to your proprietary source code. Many Macintosh users also enjoy customizing interfaces to suit their personal preferences.

All this is only possible if you make sure that your application places all its language-dependent strings into resources — any strings "hard-wired" into your source code are not accessible with a resource editor. Prototyper 3.0 places all the strings used in your interface design into STR# (string-list) resources automatically.

Loading Resources

Resources loaded into a prototype do not embody the original functionality with which they were associated in the original application. For example, when the resources for a window with a Cancel button are loaded into a prototype, the window's physical attributes and those of the Cancel button are loaded, but the ability of the Cancel button to close the window is not loaded, since that functionality was provided by the application's source code. However, such interactions can be specified in your prototype after you have loaded the resources.

Dialogs are the easiest windows to load, because each dialog includes a dialog item list resource, or DITL, which identifies the various components (e.g. buttons) included in that specific dialog. When you load in three different dialogs, Prototyper uses the DITL to decide which buttons belong to which dialog, and sets up the zones appropriately. However, there is no corresponding "window item list"; if you load a window resource, Prototyper can't determine if any other items loaded are associated with the window in the original application. For example, Prototyper can't tell which window is the rightful owner of any pop-up menus that are loaded into your prototype, and so puts them into the Menu Editor for you to deal with.

Some zones supported in Prototyper's Worksheets, such as lines or rectangles, are not associated with resources. Many times, the developer of an application associates these zones with user item resources. User items also allow the programmer to capture mouse presses within any area of a window, then react as necessary. If a window loaded into Prototyper contains user items, Prototyper creates rectangle zones as surrogate user items. (You can create user items of your own in Prototyper with the new HotSpot zones.)

Once you have loaded "raw" resources into a prototype document and tidied things up initially, you can use Prototyper's Export and Import capabilities to move all the attributes of your window, dialog or alert to other prototype documents. See the chapter Creating Windows for more information.

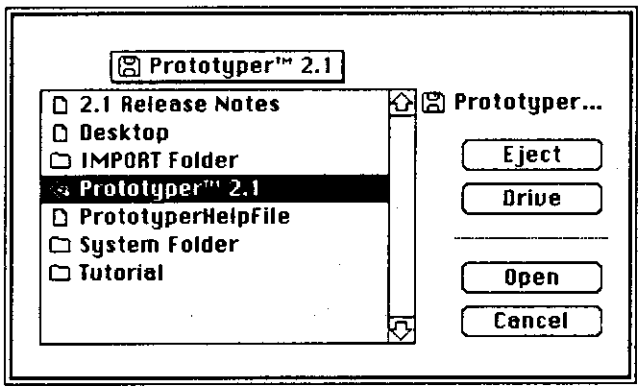
To load resources, you must have a prototype document open to receive the resources. Resources can be loaded into a new or existing prototype at any time. When you close a prototype, any resources you have loaded, but not used, are purged from the prototype.

To load resources:

1. **Open a prototype.**

   If you don't have a prototype open, choose Open from the File menu to open a new or existing prototype.

2. **Choose Load Resources from the Special menu.**



   A standard directory dialog box appears, containing all the applications and Prototyper files on your current disk.

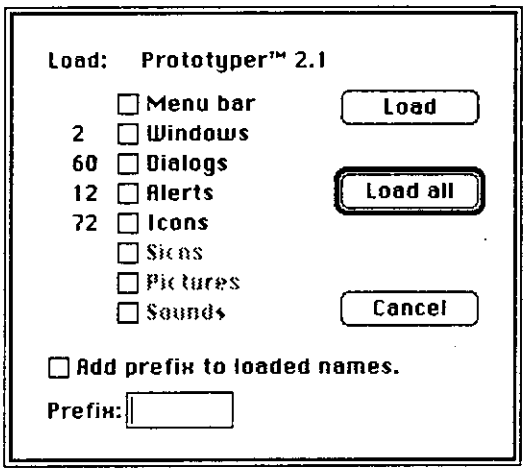
3. **Select the file you wish to load resources from.**

   Selecting a file to load from causes a dialog to appear, listing relevant resource types contained within the resource fork, such as windows, dialogs, and icons. The number of resources of each type is also shown. You have the choice of selecting one or more resource types to load, or loading all the resources available.

4. **Click the desired resources to load.**

   If you select only one kind of resource, Prototyper loads all resources of that kind. For example, if you select windows only, and there are six windows in the resource file you have opened, Prototyper loads all six windows into your prototype.

   Windows are listed in the Prototype Window, and can be opened and modified using the Worksheet. Pictures, are stored in the Picture Grabber, icons and small icons in the Icon Grabber, and sounds can be accessed thru the Link dialog.



5. **Enter a prefix to distinguish the resources from others you have loaded or created.**

   When loading resources from several resource files, you may wish to distinguish between each set of resources added to your prototype. By selecting the Add Prefix to Loaded Names option, you can specify a string prefix to add to the names applied by Prototyper to the loaded resources.

6. **Click Load or Load All.**

   The Load button loads all resources of the type or types you have selected. The Load All button loads all available resources, ignoring any type selections you may have made.

   When resources are loaded, they are converted to data in your prototype document - the prototype document itself does not contain any resources. If you wish to produce resources from your prototype document, you can use Prototyper's Code Generators to generate them.

Printing

Printing Menus and Windows

Prototyper can print menus and windows from your prototype for documentation purposes. To print a window, the Worksheet for the window must be open. If no Worksheet is open, the menus for your prototype are printed.

To print the prototype:

1. **Save the document.**
2. **To print a window, open its Worksheet. To print the menus, close all Worksheets.**
3. **Choose Print from the File menu.**

   A dialog box appears with default printing specifications.
4. **Click Print or press return**

Printing to a document

You may wish to print to a document, rather than to a printer. This lets you use your favorite paint or draw program to edit the image and set it up for printing.

To print to a document:

1. **Choose Print to file from the File menu.**

   A standard Save dialog appears.
2. **Enter a name for the document.**
3. **Click Save or press return.**

Generating Code

7. Code Generation

Generating Source Code 7-2
- Saving a prototype 7-3
- Opening the source code generator 7-3
- Selecting a target development system 7-4
- Generating source code and resources 7-5
- Compiling Generated Source Code 7-7
- Using THINK Pascal v3.0 by Symantec 7-8
- Using THINK C v4.0 by Symantec 7-13
- Using MPW Pascal v3.1 by Apple Computer, Inc. 7-18
- Compiling with MPW C v3.1 by Apple Computer, Inc 7-22

Formatting & Other Code Generation Options 7-25
- Pascal Code Generator 7-25
- C Code Generator 7-28

Trouble Shooting 7-31
- Syntax Errors 7-31
- "Code Segment Too Large" errors 7-31
- Link Errors 7-31
- "File Not Found Errors" 7-32
- Regeneration Problems 7-32
- Type Checking Problems in THINK C 7-32
- Windows with identical names 7-32
- Logic Errors 7-33
- Alerts 7-33
- Crashes 7-33
- Default Buttons 7-33
- Demo Windows don't appear 7-33

Regenerating Source Code 7-34
- Regenerating source code with THINK C or Pascal 7-35
  - Replace Core Source Files 7-35
  - Replace Core Resource Files 7-35
  - Discard new User Resource File 7-36
  - Update User Source Files 7-36
  - Recompile your Project 7-36
- Regenerating source code with MPW C or Pascal 7-38
- Trouble Shooting Regeneration Problems 7-40

Code Generation

Not only is Prototyper a versatile tool for constructing user interface prototypes, but it can quickly turn your prototypes into applications using a standard Macintosh development system. Prototyper generates Pascal and C code (called source code) and Macintosh resource files for your prototype. These files can in turn be compiled by any of the popular Macintosh Pascal or C development systems.

To use Prototyper's code generation feature, you need one of the supported development systems. The development system compiles and links the source files from C or Pascal into the only language that the Macintosh understands, 68000 machine language (object code); it then combines the object code with the resources generated for the prototype, resulting in a "stand-alone" application.

You can examine the source code Prototyper generates for your prototype, add additional code, then recompile it to make a more functional application. The improved source code architecture supported by Prototyper 3.0 even lets you go back and change aspects of the user interface of the prototype, then regenerate fresh source code without losing your modifications.

This chapter is divided into the following sections:

Generating Source Code — A tutorial on saving your prototype, loading the prototype into a Code Generator, and generating source code and resources for the prototype.

Compiling Generated Source Code — An easy-to-follow guide for each development system, giving step-by-step instructions on how to compile the code you generated in section 1. No programming experience is necessary to follow the instructions in this section for creating a stand-alone, double-clickable application.

Formatting and Other Code Generation Options — How to adjust the indentation style of generated source code to your personal preference, and how to use other options in the code generation process.

Trouble Shooting — This chapter lists problems you may encounter when compiling generated source code, and offers a simple fix for each one.

Regenerating Source Code — Simple instructions on how to regenerate source code for a previously built Application after using Prototyper to make changes to the interface.

You will also want to refer to the chapter below:

Inside Generated Source Code — Prototyper's generated source code now supports direct 'hooks' to MultiFinder services, Printing, Opening and Closing Files. This section catalogs the structure of the generated files, making it easy to add your own code to the generated application.

Generating Source Code

If you're not a proficient programmer, don't worry — creating a standalone, double-clickable application is very simple with Prototyper. The only requirements are a prototype, a development system, and the appropriate code generator. The code generators included with Prototyper 3.0 support the following Macintosh development systems: THINK's Lightspeed C™, THINK's Lightspeed Pascal™, MPW C™ and MPW Pascal™. These development systems are not included with Prototyper.

Creating a stand-alone application is as easy as following these 7 steps:

1. Save your prototype and quit Prototyper.
2. Open the source Code Generator, then open the prototype document.
3. Select your development system and generate source code.
4. Quit the Code Generator.
5. Open your development system.
6. Load the generated source code into the development system.
7. Compile the source code to create a new stand-alone application.

This section, "Generating Code, Step By Step", describes steps 1 – 4 in detail. "Compiling Generated Code" details steps 5 – 7 for each particular development system.

Saving a prototype

If you regularly use MultiFinder and have enough memory (4 megabytes or more), you can keep both the Code Generator and Prototyper open at the same time.

Your prototype document should be saved in Prototyper and closed before opening it with the Code Generator. If you don't close the prototype document before generating code, any changes you may have made to the prototype since last saving it won't be reflected in the code you generate.

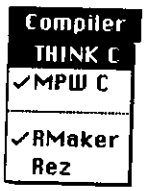
Opening the source code generator

Prototyper comes with two Code Generators, one for C development systems, and one for Pascal development systems. Select the one corresponding to the type of development system that you use. Double clicking a prototype document won't open either of the code generators; instead it opens Prototyper. The Code Generator must be opened directly.

Once opened, the Code Generator presents an "Open File" dialog, allowing you to select a prototype document to open for code generation.

Selecting a target development system

Whether you are using the C or Pascal Code Generators, you must first select the target development system from the Compiler menu.



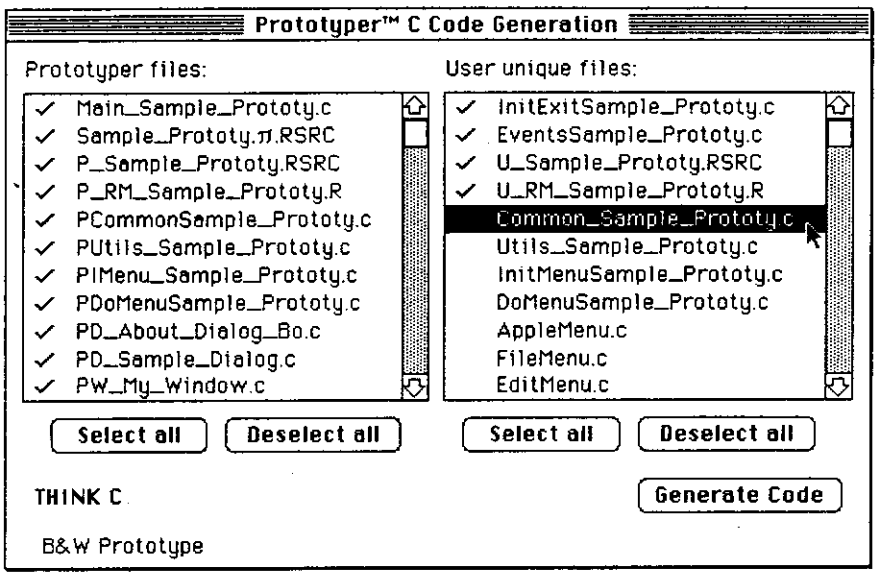
The Compiler menu also offers a choice of two types of text resource output. You may generate textual resource files in either Rez or RMaker format. You may prefer not to use resource text files; instead you could use ResEdit to create or edit compiled resources. If you don't use text resource files, you need not choose either option.

Note: If you are generating for MPW C or Pascal, the remainder of this Tutorial assumes you have selected Rez output.

The Code Generators also let you alter the formatting of the source code generated, described in more detail later in this chapter. The default settings format the code in a relatively standard manner.

Generating source code and resources

For the first generation of code for a given prototype, all files in both the "Prototyper Files" and "User Unique Files" lists must be checked. By default, items in the left list ("Prototyper Files") are checked and items in the right-most list ("User Unique Files") are not checked. Select the rightmost list is by pressing the "Select All" Button beneath it.



Now that the files are selected, press the 'Generate Code' button to start. You are presented with a standard Save dialog, asking where you would like to save the BuildMap file. This location determines where the "Proto" folder will be created, containing all the generated source files. As each file is generated it is unchecked in the appropriate list.

Note: Don't save the BuildMap file in a folder containing previously generated source code for another Prototype. If the Code Generator encounters an existing "Proto" folder inside the folder you select, it assumes you wish to regenerate source code and creates a "Proto #1" folder instead. This isn't a major problem, but you may have difficulty remembering which code is for which prototype, especially when regenerating code. Also, you will have to replace an existing BuildMap file, which may complicate updating the older project.

Prototyper's Code Generators will never overwrite a previous folder for safety's sake; you can therefore end up with a series of "Proto #2, "Proto #3", etc folders. To keep things simple, keep a separate folder for each project.

If the "Show code being generated" option has been selected in the Preferences Menu, a window is displayed during code generation, showing the source code scrolling by as it is created.

```
void DoGrow(whichWindow)
WindowPtr    whichWindow;                                          /* Handle a window bein
{
Rect    OldRect;
Point    myPt;                                                     /* Window rect be
Rect    GrowRect;                                                  /* Point for trac
long    mResult;                                                  /* Set the grow bo
                                                                   /* Result from th

if (whichWindow != NIL)                                           /* See if we have a
    {
    SetPort(whichWindow);                                        /* Get ready to draw

    myPt = myEvent.where;                                        /* Get mouse position
    GlobalToLocal(&myPt);                                        /* Make it relative *

    OldRect = whichWindow->portRect;                            /* Save the rect before

    SetRect(&GrowRect, 4, 4, (screenBits.bounds.right - screenBits.bounds.left)-4,  (screenBits
    mResult = GrowWindow(whichWindow, myEvent.where, &GrowRect);/* Grow it */
    SizeWindow(whichWindow, LoWord(mResult), HiWord(mResult), TRUE);/* Resize to result */

    switch (GetWRefCon(whichWindow))                            /* Do the appropiate wind
        {
        case Res_W_My_Window:
            Resized_My_Window(&OldRect, whichWindow);          /* Resized this window */
            break;
        default:
                                                                   /* Handle othe
```

Press and hold mouse button to pause listing...

Choose Quit from the File menu to return to the Finder.

Compiling Generated Source Code

You have now generated a complete set of source files that can be loaded into your chosen development system. The following sub-sections describe how the source files are loaded and compiled using each specific development system. Each sub-section comprises a step by step description of how to load the generated source code into your development system, and how to compile it into a stand-alone, double-clickable application.

Note: If you experience problems building a stand-alone, double-clickable applications that aren't covered by the documentation for your chosen development system, please refer to the 'Trouble Shooting' section.

Move directly to the section below that describes your chosen development system.

- Using THINK Pascal
- Using THINK C
- Using MPW Pascal by Apple Computer, Inc.
- Using MPW C by Apple Computer, Inc.

Using THINK Pascal v3.0 by Symantec

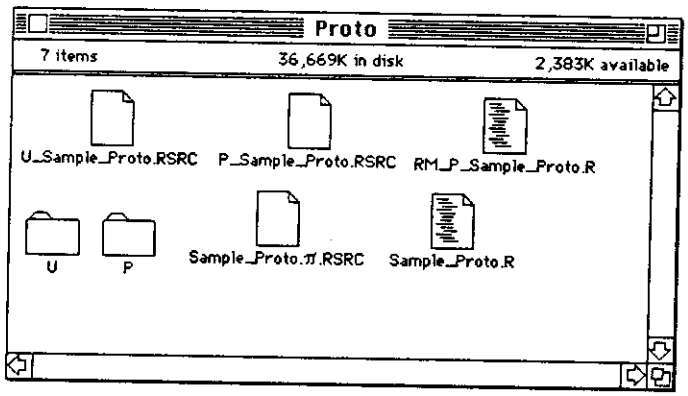
The source code generated for THINK Pascal is compatible with version 3.0. If you have an earlier version of THINK Pascal you should contact Symantec for upgrade information. The generated source code should be compatible with versions earlier than 3.0, but SmethersBarnes cannot guarantee this compatibility.

Note: If you are using an earlier version, you may need to use different libraries than those specified below, and you may also have to customize generated uses statements with the "Customize Uses Statements" option in the Code Generator's Preferences menu before you generate source code.

To compile your prototype in THINK's Lightspeed Pascal:

1. **Double-click the generated source file BuildMapxxxx.p in the Finder.**

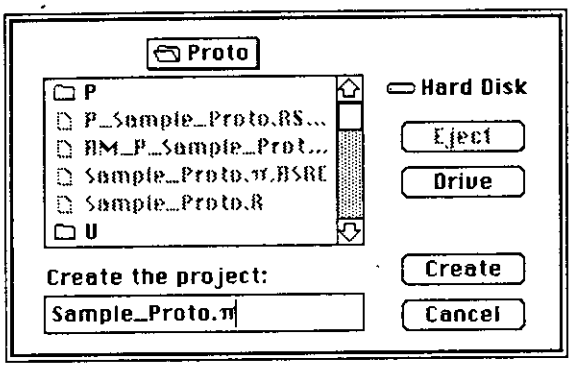
   From the Finder, open the folder you placed the "BuildMapxxxx.p" (Where xxxx indicates the name of the prototype) file in and double-click on it to open THINK Pascal.



2. **Create a new project.**

   In THINK Pascal, create a new project (by selecting "New Project" from the Project Menu), name it (whatever you choose) and place it in the "Proto" Folder. The standard THINK libraries are automatically included in the project.

   Note: THINK Pascal requires the project be in the same folder as the resource file.



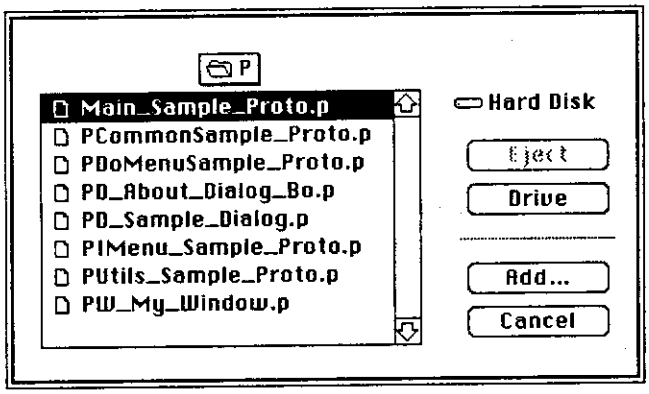
- The BuildMapxxxx.p window opens.

3. **Choose Add File... from the Project menu.**

   In the order listed in the BuildMapxxxx.p window, add each source file to the project with the Add... command.

   Within the "Proto" folder there are two sub-folders named "U" and "P". All the files from both folders must be added.

   1. **Select each file name.**



   2. **Click Add...**

      Repeat this process to load all source files into the Pascal project.

   3. **Click cancel after adding all files.**

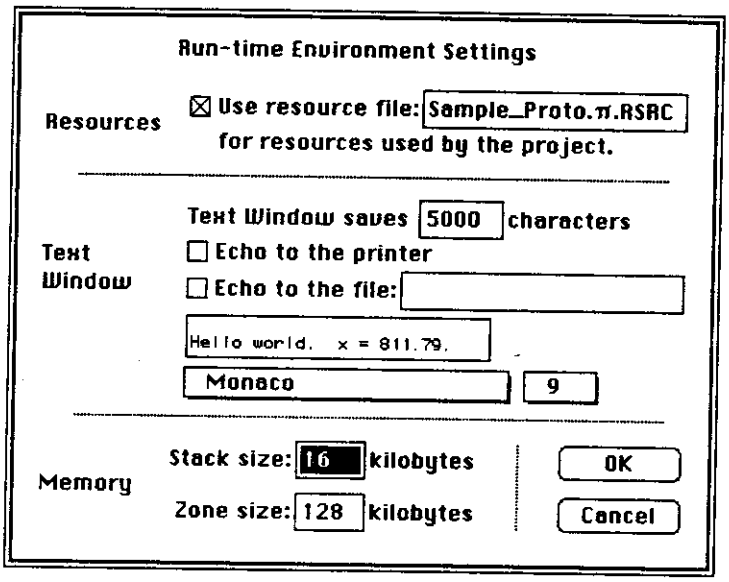
4. **Now add the resource file to the project.**

   1. **Choose Run Options from the Project menu.**
   2. **Click the check box for "Use resource file".**
   3. **Select the file "Name of your Prototype".RSRC.**

There should be three resource files to choose from. The files have similar names: one is preceded with a "P_" and another is preceded with a "U_". Ignore the P_ and U_ files, adding the third file.

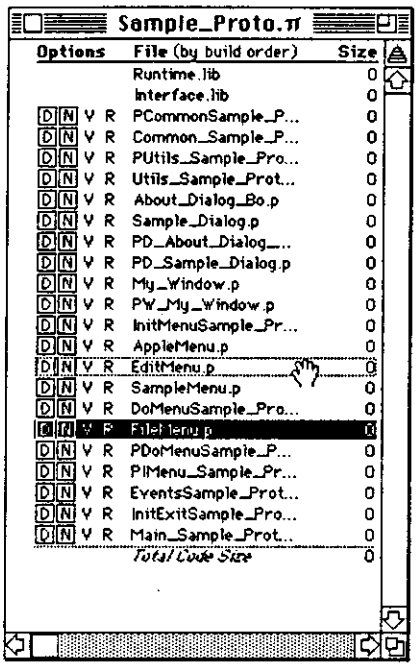
Note: If you are compiling an especially large prototype, you may need to adjust the amount of memory it uses. The default application zone size is 128k, you can raise it here.

4. **Click the Use button.**
5. **Click OK.**



6. **Make sure the files are in the appropriate build order.**

   Compare the order of the files in the project window with the order specified in the "BuildMapxxxx.p" file. To reorder the project window to match the BuildMap description, select a file you want to move. The cursor changes to a hand, and you may drag the file to its correct location.



Note: In some cases you may find files listed in the BuildMapxxxx.p window that are not in your project. This could be because you did not add all of the files from the U and P folders, or because you may be using some features (like Sounds, or Print Dialogs) that require extra libraries to be added to your project. These "extra" files include PrintCalls.lib, Printing.p, and Sound.p. They are added the same way Prototyper files are, by selecting "Add File..." from the Project Menu. They should be located in your THINK Pascal folder, inside the Libraries and Interfaces folders.

6. **Choose Build Application... from the Project menu.**

   A standard Save As... dialog box is presented for you to name and save the application with. Be careful that the name that you choose for the application does not conflict with another file in the folder.

7. **Press the return key.**

   If no errors occur while building the application, go to step 8.

   The only error that you are likely to encounter is an alert that the application segment is too large if your prototype is particularly large. THINK's Lightspeed Pascal requires that code segments (sections of the compiled application) do not exceed 32,768 bytes each. To fix this, perform the following steps.

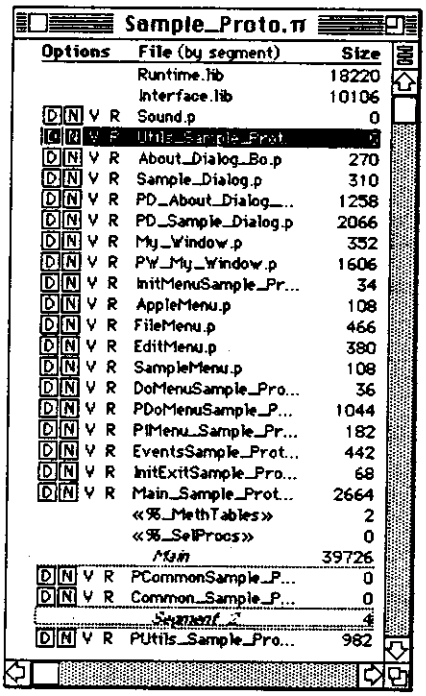
1. **Change the project view to view by segment.**

   Click on the pyramid icon on the top right of the project window. This changes the project list, viewing by segment instead of by source file name. Now you can adjust how the project uses segments.

2. **Divide the source files up into smaller segments**

   Grab files by selecting them with the mouse and dragging them down into new segments, until no segment is larger than 32,768 bytes.

   There is no need to worry about the file order while in the segment view. Grab any files you wish and move them into as many segments as you like. Just make sure no segment exceeds 32,768 bytes when you are finished.



3. **Go back to step 6.**

   Now the application can be rebuilt.

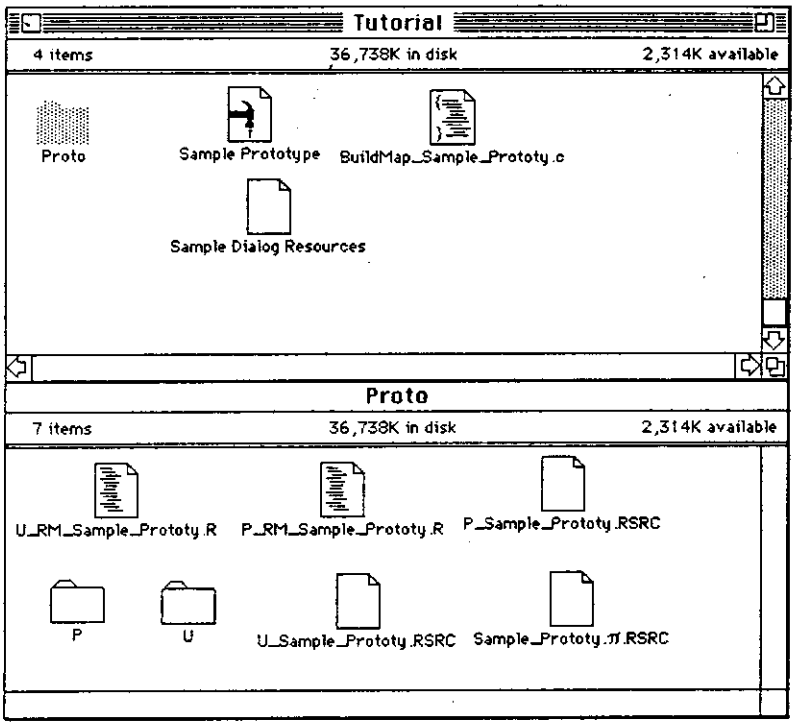
8. **Choose Quit from the File menu.**

   The application has now been created. There should be a stand-alone application in the source file folder, which implements your original prototype.

Using THINK C v4.0 by Symantec

The Prototyper 3.0 Code Generator generates source code for THINK C version 4.0. If you have an earlier version of THINK Pascal you should contact Symantec for upgrade information. The generated source code should be compatible with versions of Lightspeed C earlier than 4.0, but SmethersBarnes cannot guarantee this.

Note: If your version of THINK C is earlier than 4.0, you may need to use different libraries than those specified below, and you may also need to customize the generated #include statements with the options in the C Generator's Preferences menu before you generate source code.

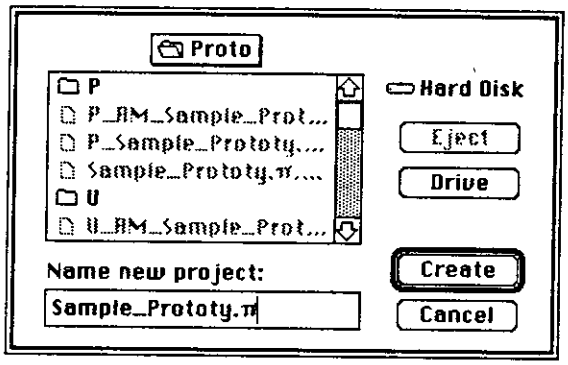


To compile your prototype with THINK C:

1. **Double-click the file BuildMapxxxx.c in the Finder.**

   THINK C opens, and presents a dialog allowing you to open an existing project or create a new one. Create a new project inside the Proto Folder. Give the project the same name as your prototype. This ensures that THINK C will open the correct resource file when you create the application.

   1. **Click the New button**
   2. **Enter the name of your Prototype**
   3. **Press the 'Create' button**

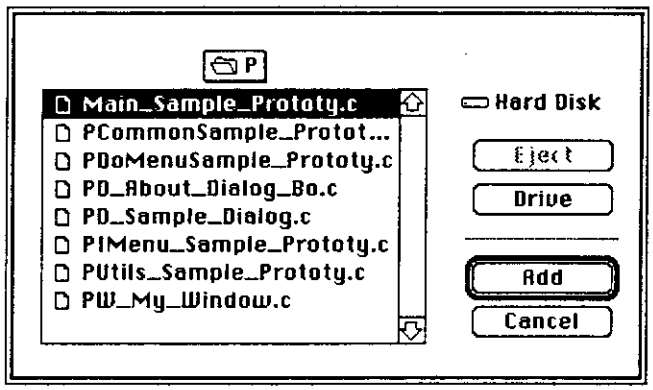


2. **Choose Add... from the Source menu.**

   Note: Be sure to select "Add..." from the bottom of the "Source" menu, rather than the "Add" command at the top of the menu. Both menu options allow the addition of source files and libraries to the project, but "Add..." makes it much faster to add multiple files.

   1. **Add all the Prototyper generated files from the U and P folders.**

      For speed, you can simply press Return repeatedly until all source files have been added. THINK removes files already added to the project from the list each time the dialog reopens.



2. **Add the "MacTraps" library to the project**

   MacTraps is located inside the "Macintosh Libraries" folder, within your THINK C folder.

3. **Click Cancel after adding all files**

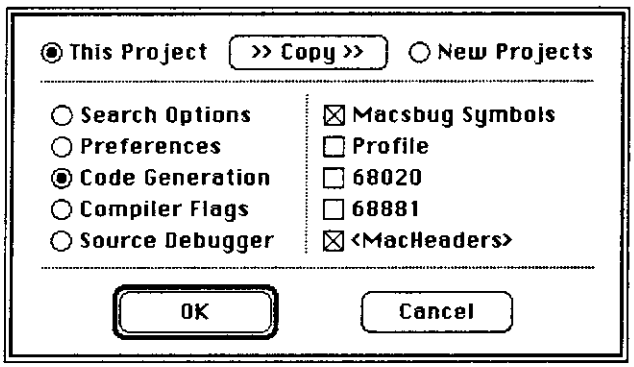
   The project is now complete.

3. **Choose Options from the Edit menu.**

Set the appropriate options for compilation. For speed and reliability, Prototyper generates source code assumes the use of MacHeaders and Prototype Checking.

Select "<MacHeaders>" with the "Code Generation" radio button and "Require Prototypes" with the "Compiler Flags" radio button.

Now you are ready to build your project into a stand-alone application.



4. **Choose Build Application... from the Project menu.**

The files in the project are compiled and linked. If the compilation is successful, a standard Save dialog box is presented, allowing you to save the completed application.

Enter 'Sample Application' as the application's name.

5. **Click Save.**

If you encountered no errors during compilation, proceed to step 9.

If your prototype is very large, you may encounter an alert that the application segment is too large. Code segments (chunks of application source code) must not exceed 32,768 bytes each. Lightspeed C separates segments with a horizontal line in the project window.

When selecting a file in the project window a vertical line is drawn to the left of it, identifying all files sharing its segment. When source files are first added to a project, they are automatically placed together in the first segment.

To fix this, perform the following steps:

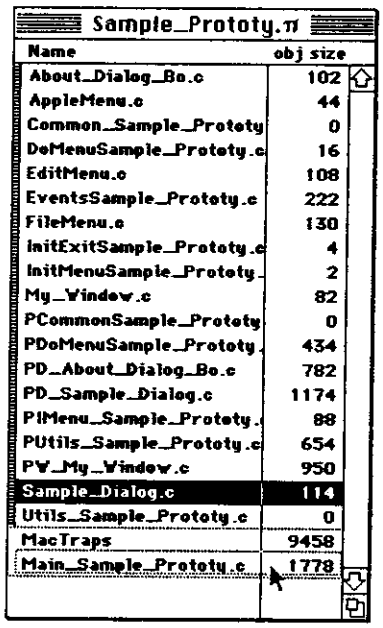
1. **Create a new segment**

   Select a file from the project window and drag it below the last file in the list. A new segment is created. A horizontal line separates it from the segment above.

2. **Divide the source files into smaller segments**

   Grab files and drag them down into new segments until no segment contains more than 32,768 bytes.

   The order of the files does not matter in THINK C — you may drag the files in any order and move them into as many segments as you like.



3. **Go back to the beginning of step 5.**

   Now the application can be recompiled.

6. **Choose Quit from the File menu.**

You have created a stand-alone application named Sample Application, which implements your original prototype. It is located in the source file folder containing the original prototype document.

Note: SmethersBarnes puts no limitations on distributing any applications created with Prototype generated code, but applications built with Symantec's THINK C are subject to any licensing agreements set forth by Symantec, and should adhere to their requirements. Prototyper is the property of SmethersBarnes, which is licensed to you and should not be given away to anyone with the stand-alone applications.

Using MPW Pascal v3.1 by Apple Computer, Inc.

Follow the installation instructions given in the MPW manual if MPW is not already installed on your hard disk (MPW requires a hard disk for development).

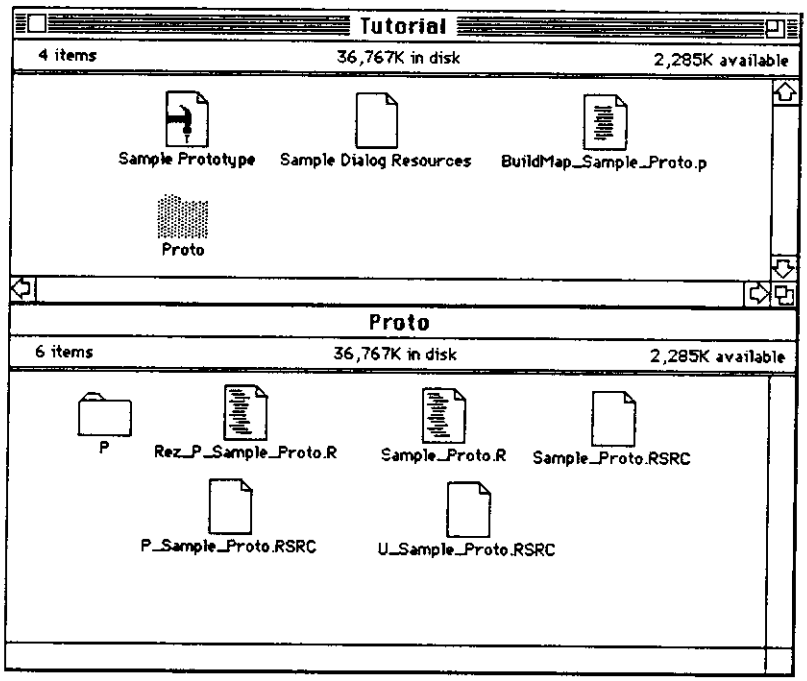
These instructions were written for version 3.1 of MPW, though other versions should work identically.

Note: If you are using an MPW version other than 3.1, you may need to customize the generated #include statements with the options in the Pascal Code Generator's Preferences menu before generating source code.

To compile your prototype with MPW Pascal:

1. **Double-click the generated file BuildMapxxxx.p in the Finder.**

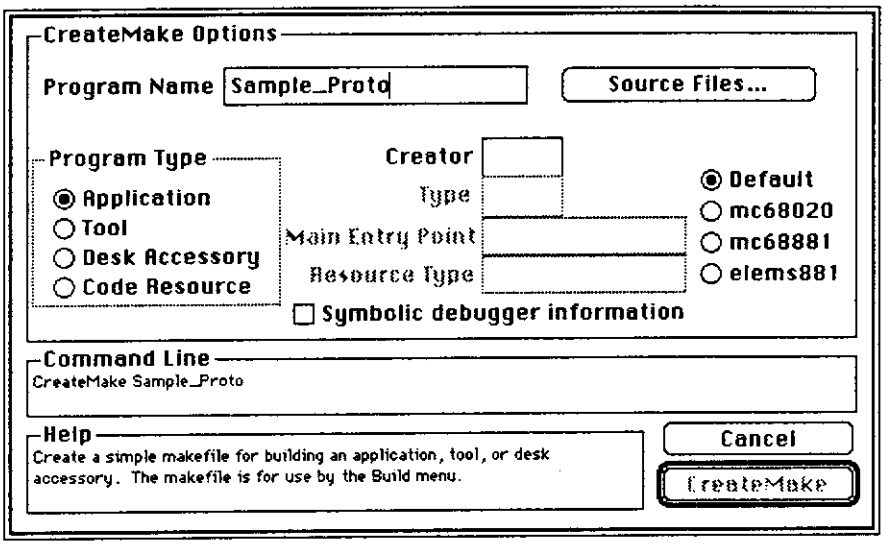
   The MPW shell opens. Double-clicking BuildMapxxxx.p (where xxxx indicates the name of the Prototype) ensures that MPW's default folder is set correctly to the folder which the generated files are located in.



2. **Select "Create Build Commands..." from the build menu.**

MPW asks for the name of the application.

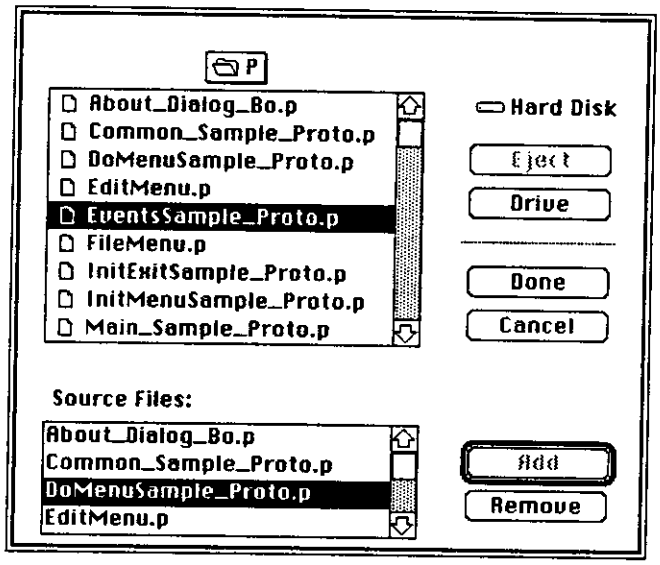
Enter "Sample Application" as your program's name, then select the 'Source Files' button.



3. **Add the Prototyper files to your project.**

Add all the files generated by Prototyper from the P folder, including the "Rez_P_(Prototype_Name).r" file placed in the "Proto" folder above the P folder.

To add a file, simply select it in the upper list, and click the "Add" button. Files that have been added are displayed in the lower list, under "Source Files". If a file is added by mistake, simply select it in the source files list, then click the Remove button to remove it from the project.

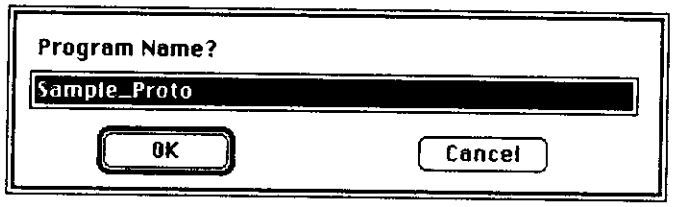


4. **Press the "Done" button.**

   The CreateMake dialog appears with the "CreateMake" button highlighted. Press the "CreateMake" button, and CreateMake will create the "make" file.

5. **Select "Build..." from the Build menu.**

   You are asked for the name of the program to build. Since you have just created a new project, its name should be entered by default. If it isn't, enter the name you used in the CreateMake window, then press Return.



MPW uses the "make" file created with CreateMake to automatically compile the application. First, it executes Rez (MPW's resource compiler), compiling the generated Rez file into resources, and loading them into your application.

The make file is then used to compile each source file using MPW's Pascal compiler, creating object files with '.p.o' name extensions.

After compilation, the Linker is called to link the separate object files together, creating the final application.

6. **Choose Quit from the File menu.**

The application has now been created. There should be a stand-alone application named Sample Application in the source file folder, which implements your original prototype. You may distribute your application as you choose.

Note: Prototyper is the property of SmethersBarnes, which is licensed to you and should not be given away to anyone with the stand-alone applications.

Compiling with MPW C v3.1 by Apple Computer, Inc

Follow the installation instructions given in the MPW manual if MPW is not already installed on your hard disk (MPW requires a hard disk for development).

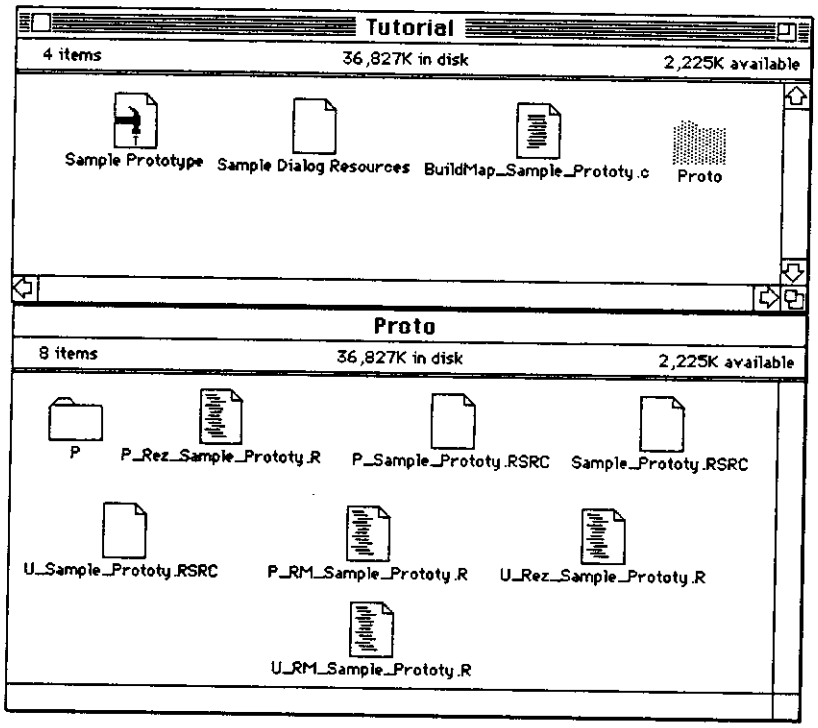
These instructions were written for version 3.1 of MPW, though other versions should work identically.

Note: If you are using an MPW version other than 3.1, you may need to customize the generated #include statements with the options in the C Code Generator's Preferences menu before generating source code.

To compile your prototype with MPW C:

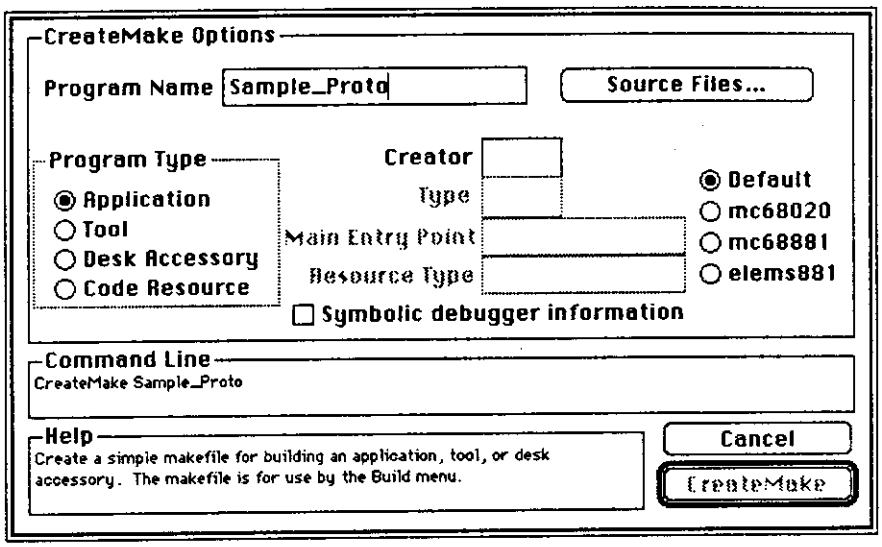
1. **Double-click the generated file BuildMapxxxx.c in the Finder.**

   The MPW shell opens. Double-clicking BuildMapxxxx.c (where xxxx is the name of the Prototoype) ensures that MPW's default folder is set correctly to the folder which the generated files are located in.



2. **Select "Create Build Commands..." from the build menu.**

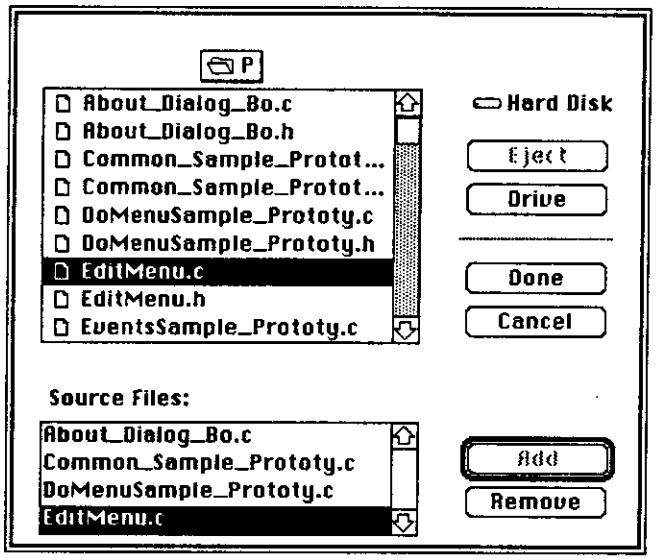
You are asked for the name of the application. Enter "Sample Application" as your program's name, then select the 'Source Files' button.



3. **Add the Prototyper source files to your project.**

Add all the files generated by Prototyper from the P folders, including the "P_(Prototype_Name).R" file, located in the "Proto" folder above both the U and P folders.

To add a file simply select it in the upper list, and click the "Add" button. Files that have been added are displayed in the lower list, under "Source Files". If a file is added by mistake, simply select it in the source files list and click the Remove button to remove it from the project.

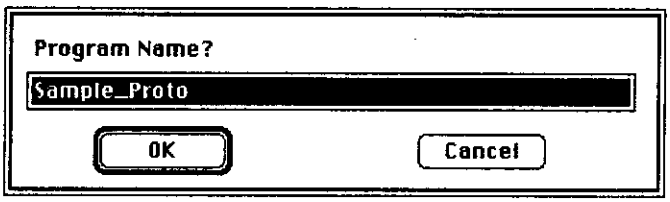


4. **Press the "Done" button.**

   The CreateMake dialog appears, with the "CreateMake" button highlighted. Press the "CreateMake" button, and CreateMake will create the "make" file.

5. **Select "Build..." from the Build menu.**

   You are asked for the name of the program to build. Since you have just created a new project, its name should be entered by default. If it isn't, enter the name you used in the CreateMake window, then press Return.



   MPW uses the "make" file created by CreateMake to automatically compile the application. First, it executes Rez (MPW's resource compiler), compiling the generated Rez file into resources and adding them to the application. The make file is then used to compile each source file using the MPW C compiler, creating object files with '.c.o' name extensions.

   After compilation, the Linker is called to link all the separate object files together and create the application.

6. **Choose Quit from the File menu.**

   The application has now been created.

Formatting & Other Code Generation Options

This section describes Code Generator options for controlling the format of generated source code. You can adjust the indentation of control loops, procedures and functions, choose between two levels of source code commenting, and display source code on-screen as it is created.

Pascal Code Generator

Compiler Menu:

1. **Think Pascal**
2. **MPW Pascal**

   These two menu options toggle, use them to select which compiler the source should be generated for.

3. **RMaker**
4. **Rez**

   These menu options select what type of text resource files are output by the generator. The menu selections aren't exclusive or required, you can select both, or neither. The compiled resource files contain all of the same resources. If you are going to use text resource files, you will need the RMaker or Rez applications to compile the files into real resources. Rez is the preferred choice, and is included with MPW as an MPW Tool or with Think Pascal as a standalone application called SARez.

Preferences Menu:

1. **Inhibit most source code comments**

   Prototyper extensively comments the source code it generates. Selecting this option reduces comments in the code to a minimum.

2. **Show Code Being Generated**

   Selecting this option displays the source code scrolling through a window as it is generated. You may pause scrolling, allowing closer examination of the code, by holding the mouse button down.

Note: This option greatly lengthens the time taken to generate source code.

3. **Text Formatting...**

The Text Formatting box offers five options for adjusting the indentation of source code. Selecting one of the five buttons opens a dialog box with sample Pascal constructs.

Pressing either of the opposing arrows to the left of a statement adjusts the indentation in or out.

Note: The indentation selected is relative: procedures are indented relative to Units, and expressions are indented relative to the procedures they are in. For example, specifying that statements within an If block be indented 6 tabs causes a statement within two If statements to be indented by 12 tabs.

Unit

Controls indenting in a generated Unit's Header block, Interface, Uses, and Implementation sections, as well as the Unit's procedures.

If...Then

The indentation of expressions and Begin and End statements within If-Then-Else constructions can be specified in this dialog box.

Procedure

Controls the indention of Const, Type, and Var declarations, as well as a procedure's Begin and End statements and expressions.

Repeat...Until

Controls indenting within Repeat and While loops, as well as the Begin and End statements for the loops.

Case

Specifies the indentation of the constant expressions, any Begin and End statements within constant expressions, and the End clause.

4. **Prototype Info...**

Displays memory available in the code generator, creation and modification date of the prototype, the prototype title, and a description of the contents of the prototype.

5. **Make All Zone Handlers.**

Zone handlers are special case statements built in windows button handler functions ("Do_Buttonxxxx()"), both in the core and stub window files. These case statements will be prebuilt with the refcons for the buttons and palettes in the window, making it easier to add code linked to the selection of specific items. However, if the window items are changed with Prototyper, and code regenerated, the old case statements will be out of date and need to be updated by hand (if user code has been added).

6. **Customize Think Pascal Uses Statements.**

Add custom uses statements to each generated pascal file with this dialog. There are 5 options.

Extra's - This is a location for adding extra system level uses statements. These statements will be the first listed after the built-in uses statements in each file.

Printing - These includes will only be added if the prototype contains printing dialogs.

Sound - These includes will only be added if the prototype contains links to sounds.

Color - Only added if the prototype has it's color option selected.

Yours - Listed last in the file's uses clause., this is the standard location for generic added uses statements.

7. **Customize MPW Pascal Uses Statements.**

Add custom uses statements to each generated pascal file with this dialog. There are 5 options.

Main - This is a location for adding extra system level uses statements. These statements will be the first listed after the built-in uses statements in each file.

Printing - These includes will only be added if the prototype contains printing dialogs.

Sound - These includes will only be added if the prototype contains links to sounds.

Color - Only added if the prototype has it's color option selected.

Yours - Listed last in the file's uses clause., this is the standard location for generic added uses statements.

C Code Generator

Compiler Menu:

1. **Think C**
2. **MPW C**

   These two menu options toggle, use them to select which compiler the source should be generated for.

3. **RMaker**
4. **Rez**

   These menu options select what type of text resource files are output by the generator. The menu selections aren't exclusive or required, you can select both, or neither. The compiled resource files contain all of the same resources. If you are going to use text resource files, you will need the RMaker or Rez applications to compile the files into real resources. Rez is the preferred choice, and is included with MPW as an MPW Tool or with Think Pascal as a standalone application called SARez.

Preferences Menu:

1. **Inhibit most comments**

   Prototyper extensively comments the source code it generates. Selecting this option reduces comments in the code to a minimum.

2. **Show Code Being Generated**

   Selecting this option displays the source code scrolling through a window as it is generated. You may pause scrolling, allowing closer examination of the code, by holding the mouse button down.

   Note: This option greatly lengthens the time taken to generate source code.

3. **Text Formatting...**

   The Text Formatting dialog offers 5 options for adjusting the source code indentation. Selecting each of the buttons opens a dialog box with sample C constructs. Pressing an arrow to the left of a statement adjusts the indentation of the statement. The indentation selected is relative, expressions are tabbed relative to the procedures they are in, and control blocks they reside in.

For example, specifying that statements within a block be indented by 6 tabs indents a statement within two blocks by a total of 12 tabs.

switch

Start brackets, the case statement, case expressions, break statements, and end brackets can all be indented in this dialog.

function

Parameter declarations, local variables, and statements can be indented in this dialog.

do...while

This dialog controls the indention of begin and end brackets, as well as the expressions in-between.

main

Parameter declarations, local variables, and statements for the main function can be indented in this dialog.

if

This dialog controls the indention of beginning and endind brackets for if statements, as well as the expressions in-between.

4. **Prototype Info...**

Displays memory available in the code generator, creation and modification date of the prototype, the prototype title, and a description of the contents of the prototype.

6. **Make All Zone Handlers**

Zone handlers are special switch statements built in windows button handler functions ("Do_Buttonxxxx()"), both in the core and stub window files. These switch statements will be prebuilt with the refcons for the buttons and palettes in the window, making it easier to add code linked to the selection of specific items. However, if the window items are changed with Prototyper, and code regenerated, the old switch statements will be out of date and need to be updated by hand (if user code has been added).

7. **Think C Color Includes**

8. **Think C Print Includes**

9. **Think C Sound Includes**

10. **Think C Extra Includes**

    Customize the standard include files for Think C generated source code with these options, these statements will be placed in "Common.h", which is included by every generated file. Color includes are only generated for Color Prototypes, Print includes if the prototype has links to printing dialogs, Sound includes if the prototype has links to sounds, and extra includes are always added.

11. **MPW C Main Includes**

12. **MPW C Print Includes**

13. **MPW C Sound Includes**

14. **MPW C Extra Includes**

    Customize the standard include files for MPW C generated source code with these options, these statements will be placed in "Common.h", which is included by every generated file. The main includes are listed first in the file. Color includes are only generated for Color Prototypes, Print includes if the prototype has links to printing dialogs, Sound includes if the prototype has links to sounds, and extra includes are always added.

15. **Your Extra Includes**

    These added include statements will be added to both Think and MPW C generated files.

Trouble Shooting

This section covers a few common programming errors, their causes, and typical solutions.

Syntax Errors

Syntax errors may occur after generated source code has been loaded into your development system. They typically occur during compilation (or build, or make, depending on your development system's terminology), causing the compilation to stop prematurely with an error message. Usually the development system opens the offending file and selects the source code line where the compiler first encountered an error.

This may not coincide with the place where the actual syntax error is, as compilers often don't notice an error until several lines later. Examine the line, and if you can't find a problem there, look at the lines immediately preceding it.

If you have problems compiling generated code, examine this list to see if your problem falls under one of these categories. If you have made functional changes to the generated source code, you need to learn how to solve the problems you will inevitably encounter.

Once you start making changes to the source code, you are on your way to becoming a Macintosh programmer, and our technical support staff will be happy to recommend the best sources of Macintosh programming information for your needs.

"Code Segment Too Large" errors

Code segments larger than 32K can be a problem with larger prototypes. Review the sub-section in "Compiling Generated Code" covering your development system for a solution. If this isn't clear, your development system documentation will offer additional help.

Link Errors

Link errors are commonly caused by omitting necessary libraries from the project. Read the section on compiling with your development system, and ensure that you have included all the correct libraries. If you are using a later or earlier version of one of the supported development systems, this could be caused by the system using different names for its libraries than in the version we document here. Usually our Technical Support staff can solve such problems in minutes.

"File Not Found Errors"

Pascal source files must be compiled in the correct order. If you don't compile in the order given, or you forget to compile a file, you will be presented with a "'UnitName' isn't in the current project, hasn't been successfully compiled, or is in the wrong build order." error from THINK Pascal. Refer to the section on compiling with your preferred development system.

Regeneration Problems

The end of the Code Regeneration Section has a trouble shooting section that lists common code regeneration problems and offers solutions to them.

Type Checking Problems in THINK C;

If the THINK C source code won't compile, make sure that you have "Check Pointer Types" turned off and "Require Prototypes" turned on in the "Options..." dialog box in the Edit menu.

Windows with identical names

Prototyper warns you when you have created windows with the same name. If you ignore these warnings and generate source code for a prototype that has identically-named windows, you will encounter syntax errors. The errors will normally be either "duplicate identifier error" in the Pascal systems, or "'WindowName' has already been declared". The simplest solution is to return to Prototyper, give all the windows unique names, and then regenerate source code.

Logic Errors

A logic error is an error in the execution of your application. The application was created correctly, but doesn't work appropriately. Sometimes the problem is minor: a window may look differently than it did under simulation. Sometimes the problem is more serious, and the application may crash.

If you have a problem in this category, the following list may offer a solution.

Alerts

If you generate code for an application, and the program you created does not link to alerts correctly, go back to the prototype and check the alert stages associated with the alert (in its Window Info window). Sometimes an alert will be staged so that it doesn't appear correctly.

Crashes

By far the most common reason for crashes or bombs in an application built with Prototyper's generated source code, is that the resource file wasn't included correctly, causing the application to be built without resources. This problem is usually indicated by the application crashing immediately upon opening.

Check to see if the resources were correctly included in the resource file with ResEdit and compare the resource fork of the application to that of the resource file (.rsrc file). The application should have all the resources that the resource file does, (as well as quite a few others). If it doesn't, you should re-examine the steps you took in building the application, paying close attention how the resource file is included.

When using THINK C, be sure that the resource file has the same name as the project, plus a '.rsrc' extension.

Default Buttons

Prototyper does not generate source code to implement default buttons in standard windows, only in dialogs and alerts.

Demo Windows don't appear

Prototyper 3.0 doesn't generate source code for demo windows. If you need sample source code for implementing demo windows, please call our Technical Support staff for assistance.

Regenerating Source Code

Prototyper's Code Generators create separate folders for User source code ("Stub" Files) and the application's main event loop ("Core" Files), allowing the source files you will most often need to augment to be kept separate from source files generated to implement the application itself. Prototyper's "Core" files contain the code to implement the user interface, the main event loop, menu handling, and window and dialog handling.

Earlier versions of Prototyper generated only core files, placing comments in the source code to indicate where to add your personal functional code. With these earlier versions, regenerating necessitated much cutting and pasting of changes between previously customized files and newly-generated files.

In Prototyper 3.0, User "stub" files contain empty functions — stubs — called from the core files, at places where you are likely to add your own functional code. These stubs do nothing other than provide a location for adding custom code, and each stub is linked to a single interface action. The stub is called by the core code in response to an event, which could be a Null Event, a mouseDown event for a particular window, a menu selection, or the opening or updating of a window. By adding code only to the stub files you can use Prototyper interactively, making changes to your Prototype, regenerating, adding new functional code, compiling, then returning to Prototyper, etc.

If you are careful to add your own source code to these User "Stub" files only, it's easy to add new features or changes to your interface design, then regenerate source code to implement the amended interface without overwriting or losing any functionality you added to previous versions of your application.

After adding functionality to your generated application, use Prototyper to make further changes to the application's interface. Feel free to change, delete or add any windows, dialogs, alerts, menus or any other items you choose. Generate code just as you did before (selecting all of the user files as well as the core files), and save the Build Map into the same folder, overwriting the old one.

Note: To make it easier to regenerate source code, it is recommended that your own source files containing functional routines be kept in an entirely different folder from both the U and the P folders. If your files are kept separate, it is much easier to compare the contents of newly regenerated U and P folders to older U and P folder versions.

Regenerating source code with THINK C or Pascal

After generating source code from the Prototype, the Code Generator creates a "Proto" folder containing the U and P folders and all the resource files. The U folder contains all the User stub files, and the P folder contains all the Core files for the Prototype.

If a "Proto" folder exists from a previous version, Prototyper creates a new folder for the generated files, called "Proto #1". Inside this folder are the newly regenerated code and resources.

Windows and Menus each have corresponding files in the new P and U folders within "Proto #1". Windows or Menus deleted in the prototype document are omitted from the new folders. Additionally, the use of some special features such as Sound or Printing in the Prototype triggers the creation of special utility files.

The following steps show how to update the original "Proto" folder with the new files to accurately reflect the new state of the Prototype.

Note: If you are using THINK C or THINK Pascal with MultiFinder, close the THINK project during this section, so that THINK won't prevent you from replacing any open files.

Replace Core Source Files

Drag the new P folder from the "Proto #1" folder into the old "Proto" folder, completely replacing the old P folder.

Caution: These are Core files. You should not change them manually, since they are replaced each time you generate code.

Note: If you have made changes to the core files despite this warning, copy your changes before replacing, and then add the changes to the new resource files.

Replace Core Resource Files

Replace the old "xxx.rsrc" and "P_xxx.rsrc" files, located in the "Proto" folder. To replace the files, simply drag the "xxx.rsrc" and "P_xxx.rsrc" files into the old Proto folder, replacing the originals.

Caution: These are Core files. You should not change them manually, since they are replaced each time you generate code.

Note: If you have made changes to the core files despite this warning, copy your changes before replacing, and then add the changes to the new resource files.

Discard new User Resource File

Once the "U_xxx.rsrc" file has been initially generated, you don't need to generate it again.

When you regenerate, leave the "U_xxx.rsrc" file unchecked in the list of files to generate. Otherwise, you can just throw away or ignore the new version after regenerating.

Update User Source Files

If you have not added to or changed any of the User stub source files in the U folder since the initial version was generated, this step is simple: just drag the new U folder into the "Proto" folder, replacing the old files.

Otherwise, open both the old and the new U folders and compare their contents.

Remove any files present in the old folder but not in the new folder, as they are no longer used by the interface (Make sure they don't contain useful code additions before throwing them away).

Move any files present in the new folder but not the old folder to the old folder. These files support new windows or menus that have been added to the interface since the original version.

Finally, throw away all the remaining files in the new U folder. These files have been previously generated, and the new copies aren't required.

Recompile your Project

THINK C

---

1. **Reopen the THINK C Project file, and open the "Build Map.c" file to display its contents.**
2. **Check the BuildMap file against the project window, and remove any files from the project window that aren't present in the Build Map.**
3. **Add any files that are present in the Build Map, but not in the project window.**
4. **Select "Make" from the "Source menu, or type "Command-M" to enter the Make dialog, and select the "Use Disk" button.**

   This forces THINK C to directly check the source files to see which have changed, and mark those to be recompiled. THINK C does not mark a file to be recompiled unless it has been modified with the THINK C editor. Select the "Make" button to recompile immediately, or select the "Don't Make" button to wait until later.

THINK Pascal

1. **Reopen the THINK Pascal Project file, and open the "Build Map.p" file to display its contents.**
2. **Check the BuildMap file against the project window, and remove any files from the project window that aren't present in the Build Map.**
3. **Add any files that are present in the Build Map, but not in the project window.**

   Make sure that all your files match the appropriate build order specified by the Build Map, reordering them if necessary (refer to the Generating Code section above for instructions on reordering the build order).
4. **When you are ready to re-compile, just type Command-G or select Go from the Run Menu**

   THINK Pascal automatically determines which files need to be recompiled.

Regenerating source code with MPW C or Pascal

MPW's Compilers do not recognize references from within a source file to files that are above it in the folder hierarchy. For this reason, Prototyper's Code Generators only create a P folder for MPW, placing all source files, core and stub, inside it.

After compiling and running a generated application, be careful to only add code or make changes to source files whose names are not preceded with a "P". The P prefix indicates the file is a Prototyper Core file, which will be overwritten if code is regenerated at a later date. The only exception to this naming convention is the Core file containing the main event loop, which is preceded by a "Main_".

Menus or Windows with names beginning with a "P" breach this convention. In this case, the User stub file for the Window or Menu is named normally, but the corresponding Core file name is prefixed with two P's. When in doubt, check the Build Map file: it always indicates which source files are Prototyper Core files and which are User stub files.

Note: To make it easier to regenerate source code, it is recommended that your own source files containing functional routines be kept in an entirely different folder from Prototyper generated folders. If your files are kept separate, it is much easier to compare the contents of a newly regenerated folder to an older folder versions. This issue is even more important when dealing with MPW's limitations.

If a "Proto" folder already exists when code is regenerated, Prototyper creates a new folder called "Proto #1", in which it places the regenerated source files and resources.

Windows and Menus in the Prototype produce corresponding files in the P folder, as well as the main event loop, etc. Windows or Menus deleted in the prototype document are omitted from the new folder. Additionally, the use of some special features such as Sound or Printing in the Prototype triggers the creation of special utility files.

The following steps show how to update the original "Proto" folder with the new files to accurately reflect the new state of the Prototype.

1. **Open the BuildMap File**

   The BuildMap file contains a list of files needed by the project in its updated form.

2. **Remove Outdated Files**

   Compare the old "P" folder with the new BuildMap file

   Throw away any files from the old "P" folder that are not listed in the BuildMap file. Throw away any corresponding object files (".o" files).

   Be careful when throwing out stub files. Make sure you didn't add any important source code to them that should be kept or moved to another source file.

3. **Add Core Files**

   Move all Prototyper Core files from the new folder into the old.

   The BuildMap file indicates which files are Prototyper Core files.

   Note: Be sure to include the "Main_" main event loop file.

4. **Add New User Stub Files**

   Move User stub files into the old folder only if they are newly generated files.

   If a file with the same name already exists in the old folder, throw away the new copy. Both files are identical when generated, but you may have added special code of your own to the old files.

5. **Update Core Resource Files**

   Replace the old "xxx.rsrc" and "P_xxx.rsrc" files located in the "Proto" folder.

   To replace, simply drag the "xxx.rsrc" and "P_xxx.rsrc" files into the old Proto folder, replacing the originals.

   Caution: These are Core files. You should not change them manually, since they are replaced each time you generate code.

   Note: If you have made changes to the core files despite this warning, copy your changes before replacing, and then add the changes to the new source files.

6. **Discard new User Resource File**

   Once the "U_xxx.rsrc" file has been initially generated, you don't need to generate it again.

   When you regenerate, leave the "U_xxx.rsrc" file unchecked in the list of files to generate. Otherwise, you can just throw away or ignore the new version after regenerating.

7. **Recompile**

   Use CreateMake (see previous section for details) to create a new Make File and to then rebuild.

   Files that have been previously compiled, and are unchanged since then will still have their object files present in the folders, and won't be recompiled by the Build command.

Trouble Shooting Regeneration Problems

Code regeneration can't work perfectly in all circumstances, sometimes you will need to make minor changes to the generated code to get it to compile and work properly. We have identified two circumstances that can cause problems.

1) Deleting Windows - When deleting window and regenerating source, some of the old stub files will still have references (#includes or uses statements) to the deleted window. These will need to be deleted by hand. Of course if you have written code that uses functions from the deleted files, it will need to be updated also.

2) Altering Menus and Palettes - The user stub files for Palettes and Menubar Menus contain Case or Switch statements that find the selected component from a selection made in the Palette or Menu. If the Menu or Palette is changed, the Case or Switch statements can become inaccurate, and will need to be updated by hand.

Inside Generated Code

8. Inside Generated Source Code

Core Files 8-2
- Core Resources 8-2
  - XXX.RSRC — Full Core compiled resources 8-2
  - P_XXX.RSRC — Partial Core resources 8-2
  - P_RM_XXX.R — Partial Core RMaker text file 8-3
  - P_Rez_XXX.R — Partial Core Rez text file 8-3
- Core Source Files 8-4
  - P_Common_XXX.Pas / P_Common_XXX.c & P_Common_XXX.h — Core Common file 8-4
  - PPrint_Common_XXX.Pas / PPrint_Common_XXX.c & PPrint_Common_XXX.h — Core common printing 8-8
  - PUtils_XXX.Pas / PUtils_XXX.c & PUtils_XXX.h— Core utilities 8-9
  - Main_XXX.Pas / Main_XXX.c — Core main program 8-11
  - P_W_XXX.Pas / P_W_XXX.c — Core window handler 8-12
  - P_IMenus_XXX.Pas, or P_IMenus_XXX.c — Core menubar initialization 8-14
  - P_DoMenus_XXX.Pas / P_DoMenus_XXX.c — Core menubar handler 8-15
  - PI_XXXMenu.Pas / PI_XXXMenu.c — Core menu initialization 8-15
  - PDo_XXXMenu.Pas / PDo_XXXMenu.c — Core menu event handling 8-15
  - P_MD_XXX.Pas / P_MD_XXX.c — Core Modeless dialog handler 8-16
  - P_D_XXX.Pas / P_D_XXX.c — Core Modal dialog handler 8-17
  - P_A_XXX.Pas / P_A_XXX.c — Core Alert handler 8-18

User Files 8-19
- User Resources 8-19
- User Source Files 8-20
  - Events_XXX.Pas / Events_XXX.c — User Event override routines 8-20
  - Common_XXX.Pas / Common_XXX.c — User common file 8-20
  - Utils_XXX.Pas / Utils_XXX.c — User stub utilities 8-21
  - InitExit_XXX.Pas / InitExit_XXX.c — User Startup and exit routines 8-22
  - XXX.Pas / XXX.c — User Window handler 8-22
  - InitMenuXXX.Pas / InitMenuXXX.Pas — User menu initialization 8-23
  - DoMenus_XXX.Pas / DoMenus_XXX.c — User menubar handler 8-23
  - XXXMenu.Pas / XXXMenu.c — User Menu list handler 8-23
  - XXX.Pas / XXX.c — User Modeless dialog handler 8-24
  - XXX.Pas / XXX.c — User Modal dialog handler 8-25
  - XXX.Pas / XXX.c — User Alert handler 8-25

Inside Generated Code

Generated code is divided into two types: high-level language source code files, and Macintosh resource files.

Source Files

The source code files generated by Prototyper 3.0 have a new structure, and are divided into Prototyper Core files and User Stub files. The Prototyper Core files are the skeleton upon which the application depends. User Stub files contain customizable (empty) stub routines called from the Core files. Simply by expanding the appropriate stubs, you can easily add functionality to the overall application. By regenerating only the Core files after a change in the structure of the prototype, the code in your user routines is preserved.

Resource Files

Resource files are a unique and essential component of Macintosh applications. Prototyper generates both "compiled" (binary) and "source code" (text file) versions of the resources. The source code versions are formatted for use with Apple Computer's RMaker, Rez, and SADeRez. The compiled versions are usually built directly into the application by the development system. If changes are needed, you can use either SADeRez, Rez, or RMaker to recompile the resource file's source code for your prototype.

Note: SADeRez is a standalone version of Rez that does not require the MPW shell to operate.

Prototyper 3.0 also allows you to customize the Code Generators by adding your own uses or include statements to all generated source files. These can be added with the Preferences menu in the Code Generators.

Note: Throughout this section files are listed using XXX to denote the name of the Prototype being generated for.

Core Files

Core Resources

Prototyper 3.0 preserves resource ID numbers for the various components of your user interface. Earlier versions of Prototyper did not preserve resource ID numbers, which complicated matters when regenerating source code after extensive functionality changes.

The generated code now includes a file called "PCommonXXX.h" or "PCommonXXX.p", which contains resource ID definitions. The ID numbers do not change between iterations. This file is included by all other source files in turn, so you can easily make any necessary ID changes in one place.

You can freely add your own custom resources to your application, as long as you use resource IDs that don't conflict with those generated by Prototyper. This can be easily done by adhering to the following ranges:

0 - 1000

Reserved for Prototyper 3.0

1001 - 32000

User resource ID numbers.

Note: The RMaker and Rez files are optional. They will only be created if selected in the Code Generator's "Compilers" menu.

XXX.RSRC — Full Core compiled resources

This resource file includes all resources used in the application, including Plugins, sounds and pictures.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_XXX.RSRC — Partial Core resources

This file contains only the Plugin, sound, and picture resources used in the generated application. These resources are not included in the Rez or RMaker text files that Prototyper 3.0 generates.

When using Rez or RMaker to compile one of the text files, copy these resources in afterwards.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_RM_XXX.R — Partial Core RMaker text file

The RMaker source file for all resources except Plugins, Pictures, and Sounds.

This file "includes" the "P_XXX.RSRC", "U_XXX.RSRC", and "XXX.R" files.

All resources are described in these other files.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_Rez_XXX.R — Partial Core Rez text file

The Rez source file for all resources except Plugins, Pictures, and Sounds.

This file "includes" the "P_XXX.RSRC", "U_XXX.RSRC", and "XXX.Rez" files.

All resources are described in these other files.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

Core Source Files

P_Common_XXX.Pas / P_Common_XXX.c & P_Common_XXX.h — Core Common file

Prototyper 3.0 creates a "Common" file that contains the resource IDs of all Prototyper 3.0 generated resources. It also contains all Prototyper 3.0 "type" declarations along with all constants and variables declared. The C Generator will place all constants in the header file.

Types

- UserEvent Record is defined.

  Prototyper defines and uses special user events to open and close Windows, Dialogs, and Alerts. In generated C code, this record is defined in header file.

Constants

- Menu list resource IDs are assigned unique menu names.
- Hierarchical menu list resource IDs are assigned to unique menu names.
- Alert resource IDs are assigned unique names.
- Modeless and Modal Dialog resource IDs are assigned unique names.
- Window resource IDs are assigned unique names.
- Popup menu resource IDs are assigned unique names.
- Icon resource IDs are assigned unique names. The highlighted icon is also given a unique name.
- SICN resource IDs are assigned unique names. The highlighted SICN is also given a unique name.
- Picture resource IDs are assigned unique names.
- Sound resource IDs are assigned unique names.

Standard Global Variables

- myEvent : EventRecord

  Event record returned by the last WaitNextEvent() or GetNextEvent call.

- WNE : Boolean

  Flag determining if WaitNextEvent() or GetNextEvent() is used by the main event loop.

  When this flag is TRUE WaitNextEvent() is used; when it is FALSE GetNextEvent() is used.

Normally, the program initialization code checks for the availability of WaitNextEvent(). You can change the value of WNE at any time to change the use of WaitNextEvent/GetNextEvent in the main event loop.

Caution: While you may use GetNextEvent() when WaitNextEvent() is available, using WaitNextEvent() when it is not available will cause a crash. Refer to Macintosh Technical Notes for full discussions of calling GetNextEvent() and WaitNextEvent().

SleepValue : Integer

Sleep value passed to WaitNextEvent() call.

Its original value is set by the program's initialization code, but it can be increased to allow more background processing time to other applications, or lowered to give the application itself more processing time.

Note: There is a bug in the original version of MultiFinder that can cause a system hang when WaitNextEvent is called with a sleep value greater than 50, so to use values greater than that, check to make sure the system version is greater than $04FF.

doneFlag : Boolean

Terminates the program. It is checked every time thru the main event loop and when it is set to TRUE the application will exit.

theInput : TEHandle

Handle to the currently active editable Text (TE) Field. If there is no active edit text field it is set to NIL.

To activate another edit text field, be sure to deactivate the current theInput, then set it to the new field and activate it. Be sure that theInput is always set to a valid edit text field or to NIL, else the application may crash when returning to the main event loop.

tempRect : Rect

Temporary rectangle, used for scratch space when drawing rectangles.

sTemp : Str255

Scratch storage for temporary string operations.

tempChar : Integer (two bytes)

Scratch storage for temporary character operations.

dataBounds : Rect

Scratch storage for temporary list operations.

cSize : Point

Scratch storage for temporary list operations.

theRow : Integer (two bytes)

Scratch storage for temporary list operations.

HasColorQD : Boolean

Flag set to TRUE at initialization time if Color Quickdraw is available.

If TRUE, windows are created as Color Windows, using GetNewCWindow() instead of GetNewWindow(), and it is safe to use Color Quickdraw calls.

HasFPU : Boolean

Flag set to TRUE at initialization time if a floating point coprocessor is installed.

If TRUE, it is safe to make direct calls to the math coprocessor to speed floating point math.

InTheForeground : Boolean

Flag set to TRUE every time the application receives a Resume event, and set to FALSE every time the application receives a Suspend event.

Can be used to determine whether the application is in the foreground or background.

UserEventList : UserEventHRec

Handle to a list of Prototyper 3.0 events

Used internally by Prototyper 3.0 code to trigger the opening or closing of windows, dialogs and alerts. The main event loop processes these events before it processes "real" Macintosh events.

typeList : SFTypeList

List used by Core routine calls to SFGetFile.

To control which types of files will be displayed in an Open File dialog, set typeList to the chosen file types — up to four types may be specified.

ErrorCode : OsErr

File I/O error number (if any) from most recent call by Core routines to SFGetFile() or SFPutFile.

Reply : SFReply

This contains the result of the user's choices from most recent call to SFGetFile() or SFPutFile(), i.e. the file's name and location, etc.

inputRefNum : Integer

outputRefNum : Integer

inputFileName : Str255

outputFileName : Str255

Reference numbers and partial path names to the currently open input and output files.

Prototyper code keeps track of the files selected in the last SFGetFile() and SFPutFile() operations as the input and output files.

Special Global Variables for Windows and Menus

Every window and menu in a prototype has a global variable declared for it.

Menu_XXXX : MenuHandle

Every menu and hierarchical menu has a MenuHandle.

The name of the menu is used as the the basis for the XXXX part of the name, and a unique number is appended to prevent duplicate identifier problems in case there is more than one menu with the same name in the prototype.

WPtr_XXXX : WindowPtr

Every menu and hierarchical menu has a WindowPtr.

The name of the window is used as the the basis for the XXXX part of the name, and a unique number is appended to prevent duplicate identifier problems in case there is more than one window with the same name in the prototype.

Special Global Variables for Global Zones

These variables are declared for all zones that have the Zone Info's Global checkbox checked in Prototyper 3.0. This enables them to maintain their values when the window they are in is closed and then reopened.

Button_Enabled_XXXX : Boolean

TRUE if currently enabled, FALSE otherwise.

For each button zone with the "Global" flag set.

Button_Ctrl__XXXX : ControlHandle

ControlHandle for a button.

HotSpot_Enabled_XXXX : Boolean

TRUE if currently enabled, FALSE otherwise.

For each HotSpot zone with the "Global" flag set.

HotSpot_Ctrl__XXXX : ControlHandle

ControlHandle for a HotSpot .

Checkbox_Enabled_XXXX : Boolean

TRUE if currently enabled, FALSE otherwise.

For each Checkbox zone with the "Global" flag set.

Checkbox_Ctrl__XXXX : ControlHandle

ControlHandle for a Checkbox .

Checkbox_Value_XXXX : Boolean

TRUE if currently checked, FALSE otherwise.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

PPrint_Common_XXX.Pas / PPrint_Common_XXX.c & PPrint_Common_XXX.h — Core common printing;

Contains global variables used to support printing.

File is only generated if the Prototype contains links to the Print and Page Setup dialogs.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

PUtils_XXX.Pas / PUtils_XXX.c & PUtils_XXX.h— Core utilities;

Contains assorted utility routines required to support the application.

The combination of routines contained in this file depends on the objects included in the prototype document. All utility routines are described here, with the conditions which trigger their inclusion.

Functions that are always added:

TrapAvailable()

Determines if a trap number is available in the current system.

GetUserEvent()

Checks the user event queue to see if any user events are pending. User events are specially created records used to open and close windows, dialogs and alerts.

Add_User_Event() or Add_UserEvent()

Adds a user event to the end of the event queue.

SetupTheItem ( )

Multipurpose function that can be used to set the title, size, associated data, enable state, maximum value and visibility of a control in a dialog or alert.

Functions added if sounds are included in the prototype:

Play_The_Sound()

Given the resource id of a sound, it loads and plays the sound.

Functions added if lists are used in the prototype:

Make_A_List()

Creates new List, and initializes it with the strings in a STR# resource.

Add_List_String()

Adds a string to the end of an existing list.

Functions added if edit text areas are used in windows in the prototype:

Get_TE_String()

Extracts the text from an edit text field (limit 250 characters).

Make_TE_Area()

Creates an edit text field with the appropriate font, style etc.

Functions added if scrollbars or gauges are used in the prototype:

HandleWScrollbar()

Handles a mouse down in a scrollbar, updating the value.

Functions added if Standard File dialogs are used in the prototype:

Do_The_Open_File()

Calls SFGetFile() and opens a selected file, storing the reference number and other result information in the appropriate globals. Returns the success of the operation.

Do_The_Save_File()

Calls SFPutFile() and creates a file, storing the reference number and other result information in the appropriate globals. Returns the success of the operation.

Functions added if printing dialogs are used in the prototype:

PageSetup()

Calls the Page Setup dialog, and stores the print settings in the global print record.

PrintJobSetup()

Fills the global print record with default settings.

OpenThePrinter()

Prepares the printing port so that subsequent drawing calls will be printed.

CloseThePrinter()

Finishes printing by spooling output, and clearing the printing port.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

Main_XXX.Pas / Main_XXX.c — Core main program

Core code implementing the application's main event loop.

Includes code to initialize the application at startup, code to cleanup before program exit, and the event handling functions.

Calls to the various window handlers change if windows are added or deleted during development of the prototype.

Initialization Code

The Macintosh ToolBox Managers are initialized.

After necessary program globals are initialized, a call is made to the user function "ApplInit_xxxx" (in the InitExitxxxx file) to allow any user application initialization code to run.

Initialization routines for all windows and modeless dialogs are called.

All windows, modeless dialogs and alerts set to open at startup are opened.

Dialog Events

At the top of the event loop a check is made to see if the event is for a modeless dialog. If it is, and the event is an update event, the update routine for modeless dialogs is called. Otherwise if it is a keyDown event it is handled directly, otherwise the Do_xxxx routine for the dialog is called to deal with other events.

Otherwise, the main event loop determines the type of event and calls the appropriate routines.

Key Event

The user stub routine HandleKey() routine is called before handling a key event, and can determine whether the character is processed or not.

Disk Event

The user stub routine HandleDisk() routine is called before the standard Disk Insertion code, and determines whether the disk insertion is handled or not.

mouseDown Event

The window in which the event occurred is determined (if any). The type of mouseDown is then determined (XXX represents the window name).

In Zoom : The Resized_XXX routine is called for the appropriate window.

In Grow : The Resized_XXX routine is called for the appropriate window.

In Drag : The Moved_XXX routine is called for the appropriate window.

In Close : The Close_XXX routine is called for the appropriate window.

In Content : The Do_XXX routine is called for the appropriate window.

In Menu Bar : The menu selection is determined and the HandleMyMenu() routine is called.

Update Event

For update events to any application window, the appropriate Update_XXX routine is called.

Activate/Deactivate Events

The Activate_XXX() routine for the appropriate window is called.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_W_XXX.Pas / P_W_XXX.c — Core window handler;

Window handler file: one is created for each window, containing all the routines handling needed operations for that specific window

(XXX represents the window name).

Init_XXX()

Initializes the window's variables at the start of the program.

Close_XXX()

Disposes of the window and releases any memory needed to support the window.

Resized_XXX()

Called for re-sizable document windows only, after the user has moved the

growbox for that window. The document window's scrollbars are updated.

Moved_XXX()

Called whenever the window is moved on the screen.

Update_XXX()

General purpose update routine that is called by an update event.

Open_XXX()

Called when the window is opened.

All zones are created by this routine, even though some may not be visible until the first update of the window.

Activate_XXX()

Called when the window is activated (selected) or deactivated.

Do_XXX()

Called for an event inside the window, such as a button being pressed.

Zone Handling

The main resource file includes a "WIND" resource for each window, and "CNTL" resources for each of the following zone types: Button, Checkbox, Radio button, Scrollbar, Plugin Button, Plugin Toggle, Plugin gauge, Button Icon, Button Picture, Button SICN, Popup menu, Palette, and HotSpots.

Buttons, Checkboxes, Radio Buttons, and Scrollbars all use the standard control definitions. All the Plugin zones use the Plugin CDEFs installed with the Plugin Installer. All other zones are supported by Prototyper 3.0 custom CDEFs.

P_IMenus_XXX.Pas, or P_IMenus_XXX.c — Core menubar initialization

Contains one routine that installs all normal and hierarchical menus in the menubar.

"MENU" resources for each menu are stored in the main resource file. These resources are created in text form for the "P_XXX.R" and the "P_XXX.Rez" files (where the XXX is the Prototype name).

Init_My_Menus()

Installs menus, then calls the user stub routine "AddExtraMenus".

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_DoMenus_XXX.Pas / P_DoMenus_XXX.c — Core menubar handler

Contains one routine that handles a mouseDown in the menubar.

Handle_My_Menu()

Determines which menu's handler routine to call; otherwise calls the user stub routine Handle_Other_Menus.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

PI_XXXMenu.Pas / PI_XXXMenu.c — Core menu initialization

PDo_XXXMenu.Pas / PDo_XXXMenu.c — Core menu event handling

"PIMenu_xxxx" loads the menus from resources and inserts them into the menu bar.

"PDoMenu_xxxx" implements links from menu selections. It handles mouseDowns in the menu bar, as well as command keys assigned to menu commands.

Menu command handling can be overridden by adding code to routines in the user menu files. A call is made to the (Do_XXXMenu()) user routine first to allow over-riding. Another call to the user routine is made after the Core code handles the menu selection. This lets you add hooks both before and after normal menu processing, for maximum programming flexibility.

DoMenu_XXX()

One routine is created for each menu, and handles selections for all items in the specific menu. Each hierarchical menu has a separate DoMenu_XXX routine. (XXX denotes the menu name).

Caution:These are Core files. You should not change them manually, since they are replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_MD_XXX.Pas / P_MD_XXX.c — Core Modeless dialog handler

The main resource file includes a "DLOG" resource for the dialog itself, and a "DITL" resource with locations for each zone of the following types: Buttons, Checkboxes, Radio buttons, Static text, Edit text, Plugin Buttons, Plugin Toggles, Icons, Pictures, SICNs, Popup menus, Palettes, and HotSpots.

Buttons, Checkboxes, Radio buttons, Static text, and Edit text all use standard control definitions. All Plugin zones use the Plugin CDEF's CNTL. Other zones are supported by Prototyper 3.0 CDEFs.

The core file contains the following standard functions, where XXX represents the dialog name:

Init_XXX()

Initializes the modeless dialog's variables at the start of the program.

Close_XXX()

Disposes of the modeless dialog, and releases any memory needed to support zones in the dialog.

Moved_XXX()

Called whenever the modeless dialog is moved on the screen, this routine calls a user stub routine to allow custom handling of actions such as moving from one screen to another, or ensuring related palettes or windows should follow a specific modeless dialog's position.

Update_XXX()

Redraws the window contents.

Open_XXX()

Called when the dialog is opened. All zones are created by this routine, even though some may not be visible until the first update of the dialog .

Activate_XXX()

Called when the dialog is either activated (selected) or deactivated.

Do_XXX()

Handles mouseDowns in the dialog.

Caution:This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_D_XXX.Pas / P_D_XXX.c — Core Modal dialog handler

The main resource file includes a "DLOG" resource for the dialog itself, and a "DITL" resource with the locations for each zone of the following types: Buttons, Checkboxes, Radio buttons, Static text, Edit text, Plugin Buttons, Plugin Toggles, Icons, Pictures, SICNs, Popup menus, Palettes, and HotSpots.

Buttons, Checkboxes, Radio buttons, Static text, and Edit text all use the standard control definitions. All Plugin zones use the Plugin CDEF's CNTL. Other zones are supported by Prototyper 3.0 CDEFs.

The Core file contains the following standard functions (XXX denotes the dialog name).

- I_PD_XXX()

  Initializes local data.

  PD_XXX()

  Opens the dialog, and handles events until it is dismissed.

  The Core file may contain the following functions, depending on which zones are contained in the dialog:

  MyFilter()

  Dialog's filter routine, created for zones that can't be directly handled by the Dialog Manager.

  Clear1RadioGroup()

  Utility function to clear a radio group.

  Refresh_Dialog()

  Update routine for zones not directly handled by the Dialog Manager

  Called from the MyFilter() routine.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

P_A_XXX.Pas / P_A_XXX.c — Core Alert handler

The main resource file includes a "ALRT" resource for the dialog itself, and a "DITL" resource with the location for zones of the following types: Buttons, Static text, Plugin Buttons, Icons, Pictures, and SICNs.

Buttons, Static text, and Edit text all use the standard control definitions. All Plugin zones use the Plugin CDEF's CNTL. Other zones are supported by Prototyper 3.0 CDEFs.

Caution: This is a Core file. You should not change it manually, since it is replaced each time you generate code.

Note: If you have made changes to this file despite this warning, copy your changes before replacing, and then add the changes to the new file.

User Files

User Resources

U_XXX.RSRC — User Partial resource file

Partial resource file containing your custom resources.

You may use this resource file to supplement generated text resource files. Use Rez or RMaker to compile one of the text files, and this file is automatically "included" by the .R and .Rez files. Convenient for resources such as pictures that aren't easily added in text form.

You can freely add your own resources to this file, but be sure to use resource IDs that don't conflict with those generated by Prototyper. This can easily be done by adhering to the following ranges:

0 -> 1000

Reserved for Prototyper 3.0 resources.

1001 -> 32000

User resource ID numbers.

XXX.R — User Partial RMaker source file;

You should not delete from or add to this file.

If you use RMaker, you can add your own resources here.

This file is not overwritten during regeneration, and is "included" by the "P_XXX.R" file.

XXX.Rez — User Partial Rez source file;

If you use Rez, you can add your own resources here.

This file is not overwritten during regeneration, and is "included" by the "P_XXX.R" file.

User Source Files

Events_XXX.Pas / Events_XXX.c — User Event override routines;

Contains four stub routines allowing you to override default event handling in four areas of the main event loop.

Most events can be better handled within a particular window's event-handling stub routine.

HandleKey()

Called from inside the event loop for special handling of key events.

HandleDisk()

Called from inside the event loop for special handling of disk insertions.

ApplLoop_XXX()

This routine is at the very top of the event loop, called right before GetNextEvent() or WaitNextEvent(). Executed each time through the event loop. A good location to add code for handling cursor changes dependent on current mouse location.

ApplEvent_XXX()

Lets you intercept any event. It also allows you to override the main event loop and ignore an event, or deal with it in a special way.

If a key event or a disk insertion event is overridden here, HandleDisk() or HandleKey() will not be called.

Common_XXX.Pas / Common_XXX.c — User common file

This common file is "used" by all other source files.

You can place global constants, types, and variables in this file and be assured that they will be available in all other stub files.

Utils_XXX.Pas / Utils_XXX.c — User stub utilities

The functions in this file can be "used" by all stub files. It offers a location for commonly used application utility routines.

Printing Dialogs

Prototyper 3.0 not only supports Print and Page Setup dialogs in source code, but also generates printing code for the Print dialog. Initially, it prints a short message, and can easily be extended.

When the Page Setup dialog is called, it updates the global variable hPrint, allocating the handle if necessary, and then copying the results of the Page Setup dialog into the print record.

When the Print dialog is called, it uses the hPrint record if allocated; otherwise, it creates a default record on its own, creates a Printing Port and draws the text message. You need only replace the text drawing with your own drawing routines to customize the output.

File Dialogs

Prototyper 3.0 not only supports SFGetFile and SFPutFile dialogs in generated code, but also generates code to open and create files.

In this file, the stubs Open_The_File() and Save_The_File() are called first. They specify the file types, and names (if needed), then call a Core routine to present the dialog. You can easily change this code to specify the file types and names of your choice.

Below are listed functions added automatically to the stub utility file:

If standard file dialogs are used in the prototype:

- Close_The_Input_File()
- Open_The_Input_File()
- Close_The_Output_File()
- Open_The_Output_File()
- Save_The_File()

If printing dialogs are used in the prototype:

Print_The_Data()

InitExit_XXX.Pas / InitExit_XXX.c — User Startup and exit routines

You can use this file to add initialization and exit routines to your application. There are two stub routines (XXX indicates the prototype name).

ApplInit_XXX()

One-time application initialization code. It is executed when the application starts.

ApplExit_XXX()

One-time application cleanup and exit code. It is executed when the application ends.

XXX.Pas / XXX.c — User Window handler

Prototyper creates a separate stub file for each window. Each stub function in the file is called for a specific window action (XXX is the window name).

U_Init_XXX()

Called when the window is initialized.

U_Close_XXX()

Called when the window is closed.

U_Resized_XXX()

Called when the window is re-sized.

U_Moved_XXX()

Called when the window is moved.

U_Update_XXX()

Called when the window is updated.

U_Open_XXX()

Called when the window is opened or selected.

U_Activate_XXX()

Called when the window is activated or deactivated.

Do_XXX()

Called when the window receives a mouseDown event.

InitMenuXXX.Pas / InitMenuXXX.Pas — User menu initialization

Executes special functions when the menu bar is created. Primarily used to add special menus at startup.

AddExtraMenus()

Called after all menus have been created in the menubar, but before calling the Toolbox routine DrawMenuBar().

DoMenus_XXX.Pas / DoMenus_XXX.c — User menubar handler

Contains a stub you can use to handle special menus. Called if a menu selection event is not for a menu belonging to your application.

The user menu handler will be called both before and after a menu selection is handled in the main loop. The user routine will know by a flag passed, whether it is being called before or after the main loop handles the selection. As usual, the user can override the handling of the menu selection by the main loop by setting a flag.

Handle_Other_Menus()

Use this routine to handle any other special menus.

XXXMenu.Pas / XXXMenu.c — User Menu list handler

A file is created for every menu containing a user routine that handles mouseDowns or command keys for the menu. A separate user stub is created for each menu, named "xxxMenu.Pas", where "xxx" is the menu title.

Do_XXXMenu()

Called before the Core menu handling code, allowing you to override Core handling of the menu selection or to perform any other application-specific operations. The routine is called again after the Core routine handles the menu selection.

XXX.Pas / XXX.c — User Modeless dialog handler

User stubs for modeless dialogs. Add code to these routines to add functionality or override the standard behavior of the Core routines (XXX is the dialog name).

U_Init_XXX()

Called when the Dialog is initialized.

U_Close_XXX()

Called when the Dialog is closed.

U_Resized_XXX()

Called when the Dialog is re-sized.

U_Moved_XXX()

Called when the Dialog is moved.

U_Update_XXX()

Called when the Dialog is updated.

U_Open_XXX()

Called when the Dialog is opened or selected.

U_Activate_XXX()

This routine is called when the Dialog is either activated or deactivated.

Do_XXX()

This routine is called when the Dialog receives a mouseDown event. It is passed the item number of the selected dialog item.

XXX.Pas / XXX.c — User Modal dialog handler

User stubs for modal dialogs.

D_Init_Modal()

Called at application startup to allow creation of any special items or special settings for items in a Dialog.

D_Filter_Modal()

Called from the dialog's filter routine, allowing the overriding of dialog events.

D_Refresh_Modal()

Called when the filter procedure handles an update event for the dialog.

D_Setup_Modal()

Called when the dialog is first created.

D_Hit_Modal()

Called for each item hit, it allows you to force an exit from the dialog.

D_Exit_Modal()

Called when the dialog is exited.

XXX.Pas / XXX.c — User Alert handler

User stub routines for Alerts (XXX denotes the Alert name).

A_Init_Alert()

Called at application startup to allow creation of any special items or special settings for items within an Alert.

Filter_Alert()

Called inside the Alert filter, and after the Alert has been responded to.

A_Hit_Alert()

Allows special responses to an alert item "hit".

Using the Plugin Installer

9. Using the Plugin Installer

Plugin Resources – An Overview 9-2
Custom Controls – 'CDEF' resources 9-2
Custom Windows – 'WDEF' resources 9-3
Custom Sounds – 'snd ' resources 9-4

Using the Plugin Installer

The Prototyper 3.0 Plugin Installer installs and removes custom control resources from Prototyper 3.0 and its Code Generators.

If a Plugin is installed in a Code Generator, but not in Prototyper 3.0, it can be matched by choosing the Match Generator to Prototyper command from the Installer's Action menu.

Individual Plugins can be removed from Prototyper 3.0 or the Code Generator by choosing the Remove from Prototyper or Remove from Generator commands from the Action menu.

All Plugins can be removed from Prototyper 3.0 or the Code Generator by choosing Remove ALL Plugins from Prototyper or Remove ALL Plugins from Generator from the Action menu.

Note: In most situations, you will prefer to install Plugins into both Prototyper 3.0 and your Code Generators

To display all Plugins currently installed in Prototyper 3.0 or a Code Generator:

1. **Choose either the Currently in Prototyper or Currently in Generator command from the Display menu.**
2. **Click the Zones, Windows, or Sounds button from the Currently in Prototyper or Currently in Generator dialog.**

To install Plugins in Prototyper 3.0 and a Code Generator:

1. **Double-click the Prototyper 3.0 Plugin Installer**
2. **Locate the Prototyper 3.0 application.**
3. **Locate the appropriate Code Generator (C or Pascal).**
4. **Select the Plugin type you wish to install from the Plugin menu.**
5. **Click the Open button, and open the appropriate Plugin from the directory dialog box.**

   Only Plugins with the correct file type and creator codes will be displayed.
6. **From the Action menu, install the Plugin into Prototyper 3.0 and the Code Generator, by choosing the Install in Both command.**

Plugin Resources – An Overview

Custom Controls – 'CDEF' resources

Controls support interaction with windows, dialogs and alerts. The zones in the worksheets of Prototyper 3.0 are implemented with control resources, for instance. Control resources are executable code, and must be written to comply with the guidelines for CDEFs documented in Inside Macintosh.

Once a custom control has been loaded into Prototyper, its icon appears in the worksheet palette along with Prototyper's standard zone tools. The custom zone tool may then be used to create zones in any Prototyper worksheet. You may also interact with the zone during Run or QuickLook simulation.

If a custom control used in a Prototyper document is also installed in a Code Generator, the control will also be supported in generated code, and becomes part of the compiled prototype application.

Each Plugin zone is loaded into Prototyper 3.0 with the Plugin Installer. The Installer can only load the Plugin if it is in the proper file format, and is accompanied by several supporting resources. The ID number of the CDEF must be between 265 and 4095.

The file's type code must be 'PZon', with creator code 'proT'. All supporting resources must have the same ID number as the CDEF, and must be included in the file for successful installation.

Supporting resource types:

- **'PROx'**

  Describes the Plugin type. It is 1 byte long. There are currently 3 possible values:

  1 specifies a button zone that is momentarily selected and released.

  2 specifies a checkbox or radio button zone, toggling between two states.

  3 specifies a gauge or scrollbar zone, with a range of possible values.

- **CDEF**

  The actual code resource for the Plugin control.

- **CDEH**

  Default pixel height of the zone (1 byte). When a new zone of this type is first created in a worksheet, its height is set to this value.

- **CDEW**

  Default pixel width of the zone (1 byte). When a new zone of this type is first created in a worksheet, its width is set to this value.

- **SICN**

  A standard SICN used in the zone's Info Window and Prototyper's Link dialog. It should be unique, and visually describe the control and its purpose.

- **STR#**

  Two lines of text describing the Plugin Zone. The first line contains the zone name. A one- or two-word description is recommended. The second line contains a short description of the zone's characteristics, used in the Zone Info dialog.

- **CNTL**

  Control template necessary for using the custom control in Dialogs and Alerts. The ProcId field of the CNTL template must be set to (16 * the CDEF ID number).

Custom Windows – 'WDEF' resources

Each Plugin Window is loaded into Prototyper 3.0 with the Plugin Installer. The Installer can only load the Plugin if it is in the proper file format, and is accompanied by several supporting resources. The ID number of the WDEF must be between 128 and 255.

The file's type code must be 'PWin', with creator code 'proT'. All supporting resources must have the same ID number as the WDEF, and must be included in the file for successful installation.

Supporting resource types:

- **WDEF**

  Code resource for the window.

- **ICON**

  Standard icon used in the New Window dialog. It should be unique, and visually represent the window's characteristics.

- **SICN**

  Standard SICN used in the Prototype Window. It should be unique, and visually describe the control and its purpose.

- **STR#**

  Two lines of text describing the Plugin Window. The first line contains the window name. A one or two word name is best. The second line contains a short description of the window's characteristics, used in the Window Info dialog.

Custom Sounds – 'snd ' resources

Sounds can be loaded into Prototyper 3.0 with the Plugin Installer, using the Load Resources command, or by another application. The Installer can only load the Plugin if it is in the proper file format. Sound Plugins are not accompanied by any supporting resources.

Note : SmethersBarnes recommends that if you are interested in using sounds in your prototype, that they be loaded from resources. This method is much more flexible than using the plugin-installer.

The file's type should 'Psnd', with creator 'proT'. Only one resource can be loaded from each file, and the resource may have any valid ID number.

- **snd**

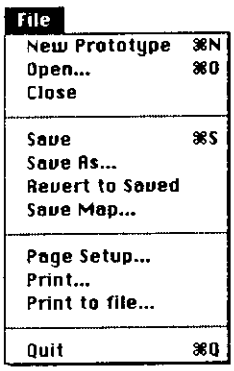
  The sound resource — its name is identical to the resource name.

Prototyper 3.0 Menus

10. Prototyper 3.0 Menus

File Menu 10-1
Edit Menu 10-2
Layout Menu 10-3
Info Menu 10-4
Special Menu 10-5
Font Menu 10-6
Style Menu 10-6

Prototyper 3.0 Menus



File Menu

New Prototype

Creates a new Prototype Window for each prototype. The New Prototype command is dimmed if a Prototype Window is open. Only one Prototype Window can be open at a time.

Open...

Opens other prototypes on the disk or hard disk. Selecting Open will display a list prototypes in the current folder or disk.

Close

Closes the active window on the screen. If the last active window on the screen is closed, it will close the entire prototype. If changes have been made, Prototyper will ask to save the changes. Clicking Yes will save the changes. If the prototype is not named, a Save as... dialog will appear allowing you to name your prototype.

Save

The Save command saves the current prototype. If your prototype is unnamed, a Save as... dialog will appear allowing you to name your prototype. If the prototype is named, changes to the prototype will be saved.

Save as...

If you have saved your prototype with a name, Save as... lets you save it under a different name. The previous version of your prototype will still be available under the old name.

Revert to Saved

Restores the last saved version of your prototype, discarding all changes made since the last save.

Save Map...

Generates a map of all of your prototype's links and dialog positions, saving it as a text document.

Page Setup...

Lets you set page size, orientation, and other options for printing windows and menus from your prototype.

Print

Prints the menus and windows of your prototype.

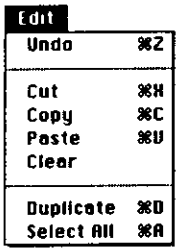
Print to file

Saves your active window as a paint file.

Quit

Quits Prototyper and returns to the Finder allowing you to save any changes made since the last save.

Edit Menu



Undo

Undoes or reverses your most recent action. Not all actions can be undone. If the action cannot be undone, the command is grayed (or disabled) and cannot be selected.

Cut

Operates on worksheet zones and items in the Menu Editor. Removes the active selection and places it on the Clipboard, replacing the previous contents.

Copy

Operates on worksheet zones and items in the Menu Editor. Copies the active selection and places it on the Clipboard, replacing the previous contents. Copy does not remove the selection from the active window.

Paste

Operates on worksheet zones and items in the Menu Editor. Places a copy of the contents of the Clipboard at the insertion point.

Clear

Operates on worksheet zones and items in the Menu Editor. Permanently removes the selection without placing it on the Clipboard.

Duplicate

Operates on worksheet zones and items in the Menu Editor. It duplicates the selected item and automatically places it at the insertion point without placing it on the Clipboard.

Select all

Selects all zones in a window worksheet.

Layout
Coordinates
Set Location...
✓Grids... ⌘G
Lock zone ⌘L
Unlock zone
Align Top
Align Bottom
Align Center
Align Left
Align Right

Layout Menu

Coordinates

The coordinate window displays the pixel coordinates of the selected zone in the worksheet. The left side of the window displays the zone's top left pixel location in the window worksheet, and the right side displays the dimensions of the selected zone.

Set Location...

Used to set the window location on the screen when the prototype is run. When selected, the active window appears inside a reduced view of the screen. The size of the window remains proportional to the size of the worksheet. To set the location, either drag the window to the desired location, or type in the coordinates, in pixels, of the corners of the window.

Grids...

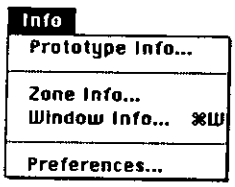
Grids may be used in the worksheet to help align zones correctly. When selected, a dialog appears allowing you to set the grid spacing in pixels, whether the zones will snap into place or not, and whether to display a visible pixel grid on the worksheet.

Lock zone/Unlock zone

Allows you to lock or unlock a zone's position and size.

Align Top, Bottom, Center, Left, Right

Allows you arrange a selected group of zones uniformly in windows, dialogs, and alerts.



Info Menu

Prototype Info...

Displays statistics about the current prototype, showing the number of windows, dialogs, and alerts in the prototype, the amount of memory available, as well as creation and modification dates and times.

Zone Info...

Lets you set additional attributes for any zone in the worksheet. When selected, a Zone Info dialog appears. The dialog can also be opened by double-clicking the active zone. The attributes of each zone type differ. See the chapter on Creating Zones.

Window Info...

Enables you to set additional attributes of the active window. You may change the name of the window, or its type. You can also designate whether the window will open when your prototype is launched. See the chapter on Creating Windows.

Preferences...

Used to set your preferences for displaying Show Additional Notes, Icons, or to drag either the actual zone or an outline of the zone in window worksheets as well as support for color prototypes.

Special
Run ⌘R
QuickLook ⌘F
Hide Prototype
Duplicate Window
Remove Window
Load resources...
Import Window...
Export Window...

Special Menu

Run

Full simulation of a prototype. When chosen, the prototype's menu bar replaces Prototyper's menu bar, and you may interact with all aspects of your prototype's user interface. To finish interacting with your prototype, choose Quit from the File menu, press Command-period, or the Escape (Esc) key to return to Prototyper.

QuickLook

Partial simulation of a prototype, simulating only the open worksheet. Prototyper's menu bar is replaced by the menu bar of your prototype. To finish interacting with your prototype, choose Quit from the File menu, press Command-period, or the Escape key to return to Prototyper.

Show/Hide Prototype

When selected, toggles between showing or hiding the Prototype Window. If the Prototype Window is open, choosing Hide Prototype hides the Prototype Window. If the Prototype Window is hidden, choosing Show Prototype displays the Prototype Window, bringing it to the front.

Duplicate Window

Duplicates the active window. When duplicated, the new window will have all of the zones and attributes of the original window. A window may only be duplicated when its worksheet is open.

Remove Window

Removes the active window from the prototype. When chosen, it permanently removes the entire window and its contents from your prototype. A window may only be removed when its worksheet is open.

Load resources...

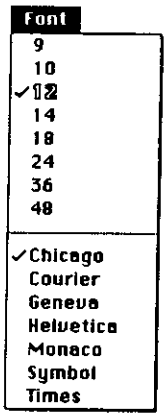
Loads resources into a prototype from another prototype document, or from any other application. A dialog appears with a list of the available documents and applications from which resources can be loaded. When you select and open the document or application from which you wish to load resources, another dialog appears. This dialog contains options for loading menus, windows, dialogs, alerts, icons and pictures.

Import Window

Imports windows from other Prototyper documents into the active prototype. To import a window, you must first export it from the original prototype.

Export Window

Exports windows from your prototype, for use in other Prototyper documents.



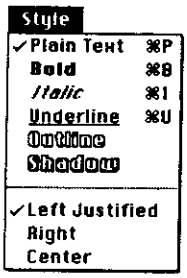
Font Menu

Point Sizes

The sizes available for the current text font are displayed in the standard fashion.

Fonts

The fonts available on your Macintosh are displayed in the standard fashion.



Style Menu

The following styles may be applied to text:

Plain Text

All text styles are removed from the text.

Bold

Italic

Underline

Outline

Shadow

Left, Right, Center Justified

Sets justification for Edit Text, Static Text and List zones to the left, right, or center.

Glossary

Glossary

Alerts

Alerts warn that an unusual situation has occurred and that data may potentially be lost if the user does not take an appropriate correcting action. There are two classes of alert signalling: beeps and alert boxes. Alert boxes come in three categories: Note Alerts, Caution Alerts, and Stop Alerts. Unaccompanied beeps are used for errors that are minor and immediately obvious. Some errors may be initially signalled by a beep, and only if the user's erroneous action is repeated is an alert box displayed to explicitly inform the user of the problem. Alert boxes resemble modal dialogs. See also Note Alert, Caution Alert, Stop Alert.

Alert Style Window

This window looks like an alert but can have all the zones of a window and acts like a window. It is not resizable or movable and has no close box.

Box Window with shadow

A plain box with no features except a shadow to give it depth. It is fixed in size and screen position. There is no close box.

Button

A button is a small screen object that is labeled with text which, when clicked or pressed, performs an instantaneous action designated by its label, such as "Cancel" or "Quit". It may also perform an action continuously when held down. If a button is the default within a dialog or alert, it is doubly outlined. This indicates visually that pressing Return or Enter is equivalent to pressing the button directly.

Check Box

Check boxes act like toggle switches. The value is "on" when the box is checked, otherwise it is "off". Check boxes do not cause any immediate action, but are used to set up conditions and let the user choose among alternatives. Check boxes appearing together in a window work independently of one another. Each check box has a name to identify its purpose.

Demonstration Window

Demo windows simulate scrolling document windows in Prototyper's simulator. There are two classes of demo window: picture and text. Demo windows can simulate a 3-D graphics image or textual data in a window, allowing the user of the prototyped application to scroll the contents, without the extensive coding necessary to implement scrolling in a full-blown application.

Dialogs

Dialog boxes are used to gather additional information from the user or provide the user with additional information about an application. There are two classes of dialog box: modal and modeless. An example of a dialog box is the window that appears when About Prototyper is chosen from the Apple menu. See also Modal Dialog Box, Modeless Dialog Box.

Document Window without Resize Box

A window without a Size box. This is a standard document window that can be dragged around the screen but cannot be resized. It can have a close box.

Edit Text

Edit text zones allow text entry by the person running the prototype. All text entered by the user is available for the prototype to operate on. You can use Cut and Paste in text entry areas.

Grids

Grids are guides to place zones in the window worksheets. Grids can make the zones snap into place, or can be used to visually align or place zones within screen grids.

Hierarchical Menu

A hierarchical menu is an extension of the standard Macintosh menu metaphor that turns a standard menu item into a submenu title. This submenu title now appears and functions just like a regular menu command, inverting when entered with the mouse, but displays a series of new menu items (or commands) when the mouse is held over it for a short time.

Icons

An icon is a small picture that is typically 32 pixels x 32 pixels. Icons are used extensively to represent ideas and to convey attributes. In Prototyper, icons can also be used like buttons to perform an instantaneous action designated by the icon. See also SICN.

Lines

Lines are horizontal or vertical inert object, and simply divide functional areas of a window visually, by guiding the user's eye appropriately. In the worksheet, lines can be changed from horizontal to vertical, or vice-versa, by dragging the zone's handle in the desired direction. The line snaps into place in the new orientation.

Menu Button

Clicking the menu button displays the Menu Editor. Use the Menu Editor to build the menu bar, menus, and commands for each prototype. The Menu Editor automatically provides the standard Apple, File, and Edit menus and their standard commands. Choose to use or delete these menus, or specify new menus and commands.

Commands can be assigned command key equivalents, disabled, assigned icons and linked to open windows.

Menu Editor

Prototyper contains a Menu Editor for building the menu bar, menus, and hierarchical menus for each prototype. It contains a Menu column, a Commands column, a hierarchical column, and various check boxes for specifying additional attributes to the commands. The Menu column contains the menu titles for the prototype. The Commands column contains the commands for each menu title. The Hierarchical column contains commands for menu item titles. See also Hierarchical Menu.

Modal Dialog Box

This dialog box restricts all interaction to itself, preventing menu selection or other windows being accessed until it is closed. Modal dialog boxes usually contain two buttons: OK and Cancel. OK closes the dialog box and performs the original command. Cancel closes the dialog box and cancels the original command. An example of a modal dialog box is the standard Open dialog box. Modal dialog boxes intrude on the free flow of user interaction — they can easily be overused.

Modeless Dialog Box

This is a dialog box that works like a window, in that it allows other windows to remain active at the same time. The user can perform other operations without closing the dialog box. A modeless dialog box is closed by clicking the close box or by choosing Close from the File menu. Buttons in modeless dialog boxes never close the dialog box. An OK button tells the computer to perform the operation, leaving the dialog box open. A Cancel button closes the dialog box.

Note Alert

This alert tells the user about a minor mistake, when no data loss is likely to occur. The user usually responds by pressing an OK button.

Picture

Pictures can be imported into a window by using the Clipboard and by using Cut, Copy and Paste. All other zones can be placed on top of pictures.

Plain Box Window

A plain box without features. It is fixed in size and screen position. There is no close box.

Plugin

A Plugin is an external objects (code resources) loaded into the prototype with the Plugin Loader. There are three types of Plugin, Custom Controls (CDEF,) Custom Windows (WDEF,) and Custom Sounds (snd.)

Pop-up Menu

A Pop-up menu is not part of the menu bar, but appears as a zone on the screen in either a window or a dialog. Pop-up menus are used for setting values or choosing from lists of similar items. When you press on this zone, the pop-up menu appears, with the current value checked and highlighted, under your cursor. An example of a pop-up menu is found in Prototyper's code generators, used when you select your preferred development system.

Prototype Window

The Prototype Window contains a menu button, a window button, a demo window button, and an area for displaying the windows created for the prototype. By clicking the menu or window button, the user builds a menu bar, or designs a number of different windows, dialog boxes, or alerts.

The window titles are displayed in the Prototype Window in alphabetical order. Prototyper defaults all window titles to "untitled", followed by a number. To edit a window, double-click the title in the Prototype Window or select the window title and choose Open from the File menu, opening the selected window.

QuickLook

Used to simulate the active window. When selected, all open windows close and the last active Worksheet appears on the screen. Prototyper's menu bar is replaced by the menu bar of the prototype. To return to Prototyper, choose Quit from the prototype's File menu, or press the Command key and type a period(.), or press the Escape key.

Radio Button

Radio buttons occur in groups, of which at any time only ONE radio button may be selected. They are called radio buttons because of their resemblance to the buttons on a car radio. Radio buttons should not cause immediate action, being used to set up conditions for a later action. Radio buttons are titled to identify what each button does.

Rectangle

Rectangles are typically used to group other zones, such as radio buttons, together visually. Rectangles may have a shadow or different outline thicknesses.

Rounded Corner Window

A box with a black title bar and rounded corners; usually used in desk accessories. It is movable and can have a close box.

Run

Menu command used to simulate a prototype, with "live" menu bar, commands, windows and zones. When chosen, the menu bar currently defined in the menu editor replaces Prototyper's menu bar at the top of the screen. You can choose commands and open windows that have been linked to those commands. Click on a button that is linked to open a window and the window appears. Select Quit from the File menu or press the Command key and type a period(.) and return to your prototype document. The exact state of the prototype is preserved during simulation.

Scroll bars

Scroll bars consist of a light grey rectangle with an arrow in a box at each end. Inside the scroll bar is a white box called the scroll box. You can create vertical and horizontal scroll bars to scroll the data within the window in either direction.

Minimum and maximum are the limits or values that the scroll bar can assume when it moves the window. The arrow step is the amount that a click in the arrow area will move the window. The page step is the amount that a click in the grey area will move the window.

Show Additional Notes

Displays help notes in the dialog boxes for windows, dialogs, alerts and zones that provide you with additional information about the type of window or zones you selected, and how a feature is generally used in an application. Prototyper is slightly faster if the additional notes are not displayed. Turn the notes on and off in the Preferences dialog.

SICN

A smaller variant of the Icon, useful in optimizing limited screen real-estate. See also Icons.

Standard Document Window

A window with a title bar and scroll bars. This is the most common type of window. It is resizable and can be dragged around on the screen. A close box is optional.

Static Text

Lines of non-editable text in windows, dialogs and alerts. If the text extends past the zone area in the window, the text will automatically flow to fit the rectangle area of the zone. Static text is usually used for titles and other static messages to be displayed inside of a window.

Stop Alert

This alert notifies the user of serious problems that may cause data loss. Immediate action is required to correct the problem.

Window Button

Clicking the window button displays a dialog box for choosing the window type to create. You are prompted to select from a number of different standard Macintosh windows, dialog boxes and alerts. Clicking on the window button always displays this dialog box preparing the user to create a new window.

Windows

Every application, whether it is a prototype for a simple desk accessory or a complicated application, will have at least one window. Windows usually display the user's data (text or graphics), or communicate with the user, such as setting the text format or notifying the user of an unusual situation. For example, a window might contain the specifications for setting up the page prior to printing, or it could contain a warning to the user about continuing the program.

Zones

Windows contain graphical objects such as icons and pictures, text areas, and components such as buttons, scroll bars, and check boxes. Some objects such as radio buttons or check boxes are used to change settings or to modify future actions.

Of these zones, buttons, radio buttons, check boxes, edit text, static text, edit text, pop-up menus, and icons are most frequently used in dialogs and alerts. Standard document windows usually contain edit text and scroll bars.

Index

A

Activate event, 8-12

Alert 2-6, 2-11, 8-18, 11-1

caution, 2-11

note, 2-11, 11-3

stop, 2-12, 11-6

Alignment

text, 3-28

zones, 3-8

B

Bottom command, 3-8

Button, 3-10, 8-14, 11-1

demo window, 1-6

enable, 3-10

help, 1-5

icon, 3-12

link, 3-11

menu, 1-4, 11-3

picture, 3-34

radio, 3-20, 3-21

set default button, 3-11

set to close dialog box, 3-11

small icon, 3-12

window, 1-5, 11-6

C

CDEF, 9-2

CDEH, 9-2

CDEW, 9-2

Center command, 3-8

Center window, 2-27

horizontally, 2-27

vertically, 2-27

Check box 3-18, 8-14, 11-1

enabled, 3-18

global, 3-18

initially checked, 3-18

Clear command 3-18, 4-20

Close box, 2-2

CNTL, 9-3

Compiling source code, 7-7

Think C, 7-13

Think Pascal, 7-8

MPW C, 7-22

MPW Pascal 7-18

Code generation introduction, 7-1, 7-2

Code regeneration, 7-34

Think C, 7-35

Think Pascal, 7-35

MPW C, 7-38

MPW Pascal, 7-38

troubleshooting, 7-40

Command keys, 4-16

Commands,

add, 4-10

cut, copy, paste, 4-9

delete, 4-7

designate, 4-16
disable, 4-12
edit, 4-19
icon, 4-18
initially checked, 4-15
link, 4-25
rearrange, 4-21
separator line, 4-13
standard, 4-4
style, 4-14
Compiler errors, 7-31
code segment, 7-31
duplicate window names, 7-32
file not found, 7-32
link, 7-31
regeneration, 7-32, 7-40
type checking, 7-32
Coordinates command, 3-4
Copy command, 4-9
Copy picture, 3-35
Core files, 8-2
Core resources, 8-2
full compiled resources, 8-2
partial compiled resources, 8-2
partial Rez file, 8-3
partial RMaker file, 8-3
Core source files, 8-4
alert, 8-18
constants, 8-4,
global variables for global zones, 8-7
global variables for menus, 8-7
global variables for windows, 8-7
main, 8-11
menu, 8-14
modal dialog, 8-17
modeless dialog, 8-16
printing, 8-8
standard global variables, 8-4
types, 8-4
utilities, 8-9
window, 8-12
zone, 8-14
Create,
command, 1-18
icon, 3-12
menu, 1-18
small icon, 3-15
window, 8-12, 2-14
zone, 3-3
Custom
controls, 9-1
C text formatting, 7-28
includes, 7-30, 8-1
Pascal text formatting, 7-26
Rez, 7-25, 7-28, 8-1
RMaker, 7-25, 7-28, 8-1
uses, 7-27, 8-1
Cut command, 4-8

D

Deactivate event, 8-12

Delete,

menus, 4-7

window, 2-28

Demo window, 2-13, 11-1

picture, 2-13

picture attributes, 2-22

textual, 2-13

text attributes, 2-23

Dialog box, 2-9, 11-2

event, 8-11

modal, 2-9

modeless, 2-10

simulate, 1-17

Disk event, 8-11

Duplicate widow 2-29

E

Edit

menu titles, 4-6

menu, 4-19

text, 3-27, 11-2

Editor,

icon, 3-12

menu, 4-2

small icon, 3-15

Event,

activate, 8-12

deactivate, 8-12

dialog, 8-11

disk, 8-11

key, 8-11

mouseDown, 8-12

update, 8-12

Export window, 2-30

F

Font

zones, 3-28

Formatting source code, 7-25

Pascal, 7-26

C, 7-28

G

Generate code, 7-2, 7-5

opening code generator, 7-3

saving prototype, 7-3

selecting development system, 7-3

Generate code button, 7-5

Grabber

icon, 3-13

picture, 3-35, 3-41

small icon, 3-15

Grid, 3-2, 11-2

command, 3-2

spacing, 3-2

Groups, radio buttons, 1-16, 3-20

H

Help file, 1-5

Hierarchical menus, 4-23, 11-2

Hot Spot, 3-31

I

ICON, 9-3

Icon, 11-2

- command, 4-18
- create, 3-12
- editor, 3-12
- global, 3-14
- grabber, 3-13
- grids, 3-14, 11-2
- link, 3-14
- location, 3-13
- set size, 3-14
- source code, 8-14
- swap, 3-13

Import window, 2-30

Initialization code, 8-11

Install,

- hard disk, 1-3
- Floppy, 1-3
- Prototyper, 1-3

K

Key event, 8-11

L

Left, 3-8

Lines, 3-33, 8-14, 11-2

Link,

- toggle, 5-2, 5-6
- button, 3-11
- check box, 3-18
- command, 1-19,4-25, 5-7
- delay, 5-2
- hotspot, 3-32
- icon, 3-14
- icon button descriptions, 5-3
- map, 5-9
- palette, 3-41
- picture, 3-36
- plugin button, 3-37
- plugin toggle, 3-38
- pop-up menu, 3-25
- print dialogs, 5-2
- radio button, 3-21
- small icon, 3-17
- sounds, 5-2
- zone, 5-8

Lists, 3-29, 8-9, 8-14

- change font, 3-30
- change point size, 3-30
- set default lists, 3-29

Load resource,

- command, 6-3

description, 6-2
introduction, 6-1
Load source code,
MPW C, 7-22
MPW Pascal, 7-18
Think C, 7-14
Think Pascal, 7-8
Load, resources, 1-13, 6-3
Location, set dialogs, 1-20
Lock zone, 3-9
Logic errors, 7-33
alerts, 7-33
crashes, 7-33
default buttons, 7-33
demo windows, 7-33

M

Map file, 5-9
Menu,
bars, 4-1
command, 1-18
create, 1-18
edit titles, 4-3
edit, 4-3
editor, 4-2, 11-3
hierarchical, 4-23
simulate, 1-20
titles, 4-5
Menus,
cut, copy, paste, 4-9
command keys, 4-16
delete, 4-7
rearrange, 4-8
standard Macintosh, 4-4
Message to be displayed,
edit text, 3-27,
static text, 3-26
Modal dialogs, 2-9, 8-17, 11-3
Modeless dialogs, 2-10, 8-16, 11-3
mouseDown event, 8-12
MPW C
compile source code, 7-22
load source code, 7-22
MPW Pascal
compile source code, 7-18
load source code, 7-18

N

Notes, show, 1-6

O

Open, 7-66
Open window, 1-8, 8-13

P

Pallette,
- editor, 3-40
- title, 3-40
- grabber, 3-41
- link, 3-41
- tools, 3-41
- code generation, 8-14

Paste, 4-9

Picture, 11-3
- copy, 3-35
- generated code, 8-14
- grabber, 3-35
- link, 3-36
- size, 3-35
- title, 3-34
- window, 2-13, 2-22

Plain box window, 2-7, 11-4

Plugin, 9-1, 11-4
- control, 9-2
- sound, 9-4
- window, 9-3

Plugin button,
- enable/disable, 3-37
- link, 3-37
- title, 3-37

Plugin gauges,
- default width, 3-39
- enable/disable, 3-39
- initial values, 3-39
- minimum and maximum values, 3-39
- stepping rates, 3-39
- title, 3-39

Plugin resources, 9-2
- types, 9-2

Plugin toggles,
- enable/disable, 3-38
- initially selected, 3-38
- link, 3-38
- title, 3-38

Pop-up menu, 3-24, 11-4

Preferences,
- C code generation, 7-5
- Pascal code generation, 7-5

Print command, 6-6

Print dialogs, 5-2, 5-6, 8-10

Print format, 6-6

Print to file command, 6-6

proT, 9-2, 9-3, 9-4

Prototyper tutorial, 1-7

Prototype window, 1-4, 11-4

PROx, 9-2

Psnd, 9-4

PWin, 9-3

PZon, 9-2

Q

QuickLook command, 1-17, 11-4

R

Radio Button, 3-20, 11-4
- enable, 3-20
- group, 3-20
- initially selected, 3-20

Rectangle, 3-31, 8-14, 11-4

Regeneration, 7-34

Remove window, 2-28

Requirements, 1-1

Resize,
- window, 2-17, 8-12
- zone, 3-5

Resources,
- core, 8-2
- load, 6-3
- user, 8-19
- what are, 6-2
- types, 7-54

Right, 3-8

Rounded corner window, 2-7, 11-5

Run, 1-20, 11-5

S

Scroll bar, 2-3, 3-22, 8-10, 8-14, 11-5
- set minimum and maximum stepping, 3-28

Select all, 3-7

Set location, 2-26

Show additional notes, 11-5

SICN, 3-15, 9-3, 11-5

Simulate,
- dialog, 1-17
- menu, 1-20
- prototype, 1-2, 1-12
- window, 1-10, 2-31

Size box, 2-2

Small icon,
- create, 3-15
- editor, 3-15
- generated code, 8-14
- grabber, 3-16
- grids, 3-17
- link, 3-17
- location, 3-16
- set size, 3-17
- swap, 3-16
- global, 3-17

snd, 9-4

Sound, 5-6, 8-9, 9-4

Standard document window, 2-5, 11-5

Standard file dialog, 5-6, 8-10

Static text, 3-26, 11-5

STR#, 9-3

Style, 3-28

T

Text,
- alignment, 3-8
- edit, 3-27, 3-28, 8-9
- static, 3-26,3-28

Toggle, 5-6

Top, 3-8

Tutorial, 1-7

U

Unlock zone, 3-9

Update event, 8-12

User files, 8-19

User resources,
- partial resource file, 8-19
- partial Rmaker file, 8-19
- partial Rez file, 8-19

User source files,
- alert, 8-26
- common, 8-20
- event, 8-20
- exit, 8-22
- menu, 8-23
- modal dialog, 8-25
- modeless dialog, 8-24
- utilities, 8-21
- startup, 8-22
- window, 8-22

W

WDEF, 9-3

Window,
- additional attributes, 2-19
- alerts, 2-11, 8-18
- behavior, 2-4
- characteristics, 2-2
- creation, 2-14
- delete, 2-28
- demonstration, 2-13
- dialog boxes, 2-9, 2-10
- duplicate, 2-29
- export, 2-30
- generated code, 8-12
- import, 2-30
- info, 2-19
- location, 2-26
- resize, 2-17
- simulation, 2-31
- types, 2-5
- worksheet, 2-16

Z

Zone, 11-6

aligning, 3-8

button, 3-10, 8-14

check box, 3-18, 8-14

coordinates, 3-4

creation, 3-3

edit text, 3-27, 8-14

grids, 3-2

hot spots, 3-31

icon, 3-12

lines, 3-33

lists, 3-29

locking, 3-9

moving, 3-6

palettes, 3-40

picture, 3-34, 8-14

plugin buttons, 3-37

plugin gauges, 3-39

plugin toggles, 3-38

pop-up menus, 3-24, 8-14

radio buttons, 3-20, 8-14

rectangle, 3-31, 8-14

resizing, 3-5

scroll bars, 3-22, 8-14

small icon, 3-15

static text, 3-26, 8-14